UNIVERSITY OF CALIFORNIA,
IRVINE

Variants of Privacy Preserving Set Intersection and their Practical Applications

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Sky Faber

Dissertation Committee:
Professor Stanislaw Jarecki, Chair
Professor Pierre Baldi
Professor Michael Goodrich

2016

DEDICATION

To Gabe, my guide in an ocean of darkness.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

ACKNOWLEDGMENTS

Throughout my time at the University of California, Irvine I met many great minds. I was shaped and molded for better or worse through my interactions with each of them. While I gained great knowledge, those interactions were more valuable than anything I learned. I am immensely grateful to have had the opportunity to be touched by all of them. I'd like to thank a few of them specifically who made this work possible with their guidance and support.

First and foremost, I'd like to thank Gene Tsudik my mentor and one-time friend. Gene taught me not only how to be an effective researcher but how to think and write critically. Thank you for including me in brainstorming sessions even when I was still a kid who knew nothing. More than anything, thank you for giving me a space to work and grow at my own rate. I would be half the man I am today without having the opportunity to work with you for so many years.

I would also like offer my wholehearted gratitude to Stanislaw Jarecki. Stas takes cryptography education seriously and with great passion. He taught me to be a real cryptographer. Without his dedication and rigor I would never have had the necessary skills to succeed in this field. On behalf of all of your students, thank you for assigning the tough problems and thoughtfully grading all of our chicken scratch proofs. Your extra curricular crypto classes inspired me to be better than I am. And, your extra long brainstorming sessions inspired me to invent something new, thank you. Most importantly, thank you for being there in my time of need. I'll never forget all that you have done for me.

I would also like to thank each of my defense committee members, Stanislaw Jarecki, Pierre Baldi, Michael Goodrich. Thank you for your support and for your ears.

Special thanks to each of my co-authors. My contributions could not shine alone. I would not be here without their thought, effort, and support. In particular, thank you to Emiliano De Cristofaro, Karim El Defrawy, Paolo Gasti, Hugo Krawczyk, and Michael Steiner for your mentorship. You're feedback was critical in my development.

I am also very grateful to each of the members of the SPROUT research group who I happily co-located with so many years: Gergely Acs, Mishari Almishar, Filipe Beato, Tatiana Bradley, Xavier Carpent, Alberto Compagno, Mauro Conti, Sargis Dudaklyan, Luca Ferretti, Paolo Gasti, Cesar Ghali, Tyler Kaczmarek, Quan Nguyen, Naveen Nathan, Ekin Oguz, Ronald Petrlic, Kasper Rasumussen, Norrathep Rattanavipanon, Marc Roeschin, Jaroslav Sedenka, Michael Steiner, and Christopher Wood. Not only were you all an endless source of inspiration my darts game would be shot without you.

In addition to all the great souls I met in my field. UCI was a place to meet lifelong friends: Kevin Bache, Kyle Benson, Jimmy Foulds, Cesar Ghali, Moshe Lichman, Ekin Oguz, Kristin Roher, Mehdi Sadri, Jonathan Shoemaker, and Patricia Sponaugle. Thank you all for listening to my late night kvetching. Our success will forever be intertwined.

I was supported during my years at UCI by fellowships from the Bren School of Information and Computer Science, as well as through several teaching and research assistantships. Many thanks goes to IBM and the IARPA SPAR program for supporting me through my education by letting me do what I love. Thanks also to Marco Levorato and Ray Klefstad for being excellent teachers and role models who truly care about their students.

Finally, I'd like to extend my deepest thanks to my parents. They continuously encouraged me to pursue my dreams and consistently pushed me towards higher education. They were always there when I needed a helping hand. I am eternally grateful.

CURRICULUM VITAE

Sky Faber

EDUCATION

Doctor of Philosophy in Computer Science — **2016**
University of California, Irvine — *Irvine, California*

Master of Science in Computer Science — **2013**
University of California, Irvine — *Irvine, California*

Bachelor of Science in Computer Science and ACMS — **2009**
University of Washington, Seattle — *Seattle, Washington*

Associates in Science — **2007**
Bellevue College — *Bellevue, Washington*

RESEARCH EXPERIENCE

Graduate Research Assistant — **2011–2016**
University of California, Irvine — *Irvine, California*

Cryptography Research Intern — **Summer 2015**
IBM Research, Watson — *Yorktown Heights, New York*

Research Intern — **Summer 2012**
HRL Laboratories — *Malibu, California*

TEACHING EXPERIENCE

TA for Network & Distributed Systems Security (ICS 203) — **Winter 2016**
University of California, Irvine — *Irvine, California*

TA for Principles in System Design (ICS 53) — **Winter 2015**
University of California, Irvine — *Irvine, California*

TA for Computer and Communication Networks (ICS 174) — **Fall 2014**
University of California, Irvine — *Irvine, California*

PAPERS IN SUBMISSION OR UNDER REVIEW

Private Projections based on Private Set Intersection **2017**
ACM Asia Conference on Computer and Communications Security

REFEREED CONFERENCE PUBLICATIONS

Bounded Size-Hiding Private Set Intersection [BFT16] **2016**
Security and Cryptography for Networks

Rich Queries on Encrypted Data: Beyond Exact Matches [FJK+15] **2015**
European Symposium on Research in Computer Security

UnLinked: Private Proximity-based Off-line OSN Interaction [FPT15] **2015**
Workshop on Privacy in the Electronic Society

Three-party ORAM for Secure Computation [FJKW15] **2015**
International Conference on the Theory and Application of Cryptology and Information Security (Asiacrypt)

Secure Genomic Testing with Size-and Position-hiding Private Substring Matching [DCFT13] **2013**
Workshop on Privacy in the Electronic Society

Genodroid: are privacy-preserving genomic tests ready for prime time? [DCFGT12] **2012**
Workshop on Privacy in the Electronic Society

ABSTRACT

Variants of Privacy Preserving Set Intersection and their Practical Applications

By

Sky Faber

Doctor of Philosophy in Computer Science

University of California, Irvine, 2016

Professor Stanislaw Jarecki, Chair

Private Set Intersection (PSI) is a cryptographic primitive that allows two network connected parties with hidden inputs to jointly compute the intersection of these inputs while keeping their specific inputs secret. PSI can be used as a building block for a variety of applications, most notably querying a remote relational database without revealing the query or the database. Many constructions of PSI exist, each building off of a subset of an assortment of cryptographic primitives such as: oblivious transfer, hash functions, garbled circuits, public key encryption and signature schemes, and basic number theoretic hardness assumptions. In this dissertation, we study variations of PSI and their practicality to modern day applications. Specifically, we study new security constraints for PSI that arise from applications such as genomics, consumer applications, and inter-agency information sharing. These constraints lead to several novel secure computation protocols. Through actualized prototypes of these schemes we conclude that, specialized PSI protocols are fast enough for use today, even on resource constrained hardware.

Chapter 1

Introduction

Recent research has unveiled the practicality of efficient Privacy Preserving Set intersection, also known as Private Set Intersection or PSI, in an assortment of security models. Since PSI's inception the research community has focused primarily on refinement of the primitive in two areas: greater efficiency (reduced cryptographic operations) and stricter privacy requirements (fewer, more innocuous cryptographic assumptions). In this thesis, we examine several variants of PSI motivated by practical applications. We detail the necessity for these variants as well as the unique privacy challenges that arise from the alterations.

The protocol constructions examined within differ from traditional PSI in several, often overlapping, ways. Specifically they have one or more of the following characteristics: modified output requirements modified matching criteria, stricter privacy guarentees, or relaxed security requirements.

In addition to Private Set Intersection we also examine the closely related problem of Secure Pattern Matching (SPM). SPM itself can be thought of as a variant of PSI with modified matching criteria and output. Each is a secure computation protocol between two parties each with private inputs. The protocol computes for one or both parties a function over

these private inputs. The protocols in this thesis are presented as a modification of either SPM or PSI depending on the requirements of the specific application.

The main contributions of this work are:

1. Design and prototype implementation of GenoDroid, a system for privacy preserving genetic testing. GenoDroid is a "cradle-to-grave" solution focusing on securing genomic data from the time it is generated by the sequence machine to time of use on user's smartphones.

2. Design and prototype implementation of UnLinked, a system for supporting proximity-based offline social network interactions. UnLinked offers authentic offline friend-of-friend discovery without the use of an actively connected central server. Perhaps most notably the UnLinked prototype seamlessly handles detection and protocol execution with physically proximate peers, notifying users only in the event of a "match".

3. Design and implementation of the finderlib framework. A Java based library for running privacy preserving cryptographic protocols on smartphones without the need for a central server. Finderlib is a central component of both GenoDroid and UnLinked.

4. Design and analysis of several novel cryptographic primitives. Briefly these are: Authorized Two-Way Private Set Intersection (ATW-PSI), Authorized Two-Way Private Set Intersection Cardinality (ATW-PSI-CA), Size and Position Hiding Substring Matching (SPH-SM), and Substring Searchable Symmetric Encryption OXT (SUB-SSE-OXT).

5. Implementation of the SUB-SSE-OXT protocol, supporting real time privacy preserving substring queries over a SQL style database with 100s of millions of rows.

In this thesis I will describe my solutions to four problems in the area of Private Set Intersection and the closely related area of Private Pattern Matching.

In this chapter we cover: the basics of Private Set Intersection and Secure Pattern Matching, the types of protocol modifications we encounter, and the practical applications our protocols are designed for.

Subsequent chapters are dedicated to a specific problem solved by a novel protocol or system design, with the exception of Chapter 4 which presents existing solutions to SPM in-depth. Specifically the problems solved in this thesis are:

1. Chapter 2 addresses the problem of using PSI to build *practical privacy preserving genomic testing on smartphones* and it is based on my publication *Genodroid: are privacy-preserving genomic tests ready for prime time?* written along with Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik [DCFGT12].

2. Chapter 3 addresses the problem of *bi-lateral private set intersection over authorized input* and it is based on my publication *UnLinked: Private Proximity-based Off-line OSN Interaction,* written along with Ronald Petrlic and Gene Tsudik [FPT15].

3. Chapter 5 addresses the problem of *size- and position-hiding private substring matching* and it is based on my publication *Secure Genomic Testing with Size-and Position-hiding Private Substring Matching,* written along with Emiliano De Cristofaro and Gene Tsudik [DCFT13].

4. Chapter 6 addresses the problem of *private pattern matching over symetrically encrypted data* and it is based on my publication *Rich Queries on Encrypted Data: Beyond Exact Matches,* written along with Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel Rosu, and Michael Steiner [FJK+15].

1.1 Private Set Intersection

Private Set Intersection (PSI) is a Secure Computation Protocol realizing the specific function of set intersection. Informally, PSI is a two party protocol in which each party holds a set they wish to keep secret and a single party learns the intersection between these two sets. Ideally, no party learns any additional information about the input sets beyond the protocol output. That is, no information about the opposing parties input is learned except what can be derived from the intersection itself. Unfortunately, this ideal functionality is difficult, if not impossible, to achieve. Instead, most authors build protocols for the slightly amended functionality that reveals the size of both input sets. This is discussed in detail in Section 1.3. Formally PSI is defined as:

Definition 1.1 (Private Set Intersection (PSI))**.** *A protocol between Server with input* $\mathcal{S} = \{s_1, \ldots, s_w\}$*, and Client with input* $\mathcal{C} = \{c_1, \ldots, c_v\}$*. At the end of the protocol execution Client learns exactly* $(\mathcal{S} \cap \mathcal{C}, |\mathcal{S}|)$*, and Server learns exactly* $|\mathcal{C}|$*. PSI securely implements:* $\mathcal{F}_{\mathsf{PSI}} : (\mathcal{S}, \mathcal{C}) \mapsto (|\mathcal{C}|, (\mathcal{S} \cap \mathcal{C}, |\mathcal{S}|))$.

In some cases learning the intersection is equivalent to learning both input sets. For example, if the two sets are identical. Due to this in some situations learning the intersection is unacceptable. In these cases it is often advantageous to learn a function over the intersection. Of particular interest is the size, or cardinality, of the intersection.

Definition 1.2 (Private Set Intersection Cardinality (PSI-CA))**.** *: A protocol between Server with input* $\mathcal{S} = \{s_1, \ldots, s_w\}$*, and Client with input* $\mathcal{C} = \{c_1, \ldots, c_v\}$*. At the end of the protocol, Client learns exactly* $(|\mathcal{S} \cap \mathcal{C}|, |\mathcal{S}|)$ *and Server learns exactly* $|\mathcal{C}|$*. PSI-CA securely implements:* $\mathcal{F}_{\mathsf{PSI\text{-}CA}} : (\mathcal{S}, \mathcal{C}) \mapsto (|\mathcal{C}|, (|\mathcal{S} \cap \mathcal{C}|, |\mathcal{S}|))$.

One can imagine even further limitations on the output information. Some works focus on generic solutions that support securely computing any function of the intersection at vastly

increased cost. One natural reduction of information from cardinality is that of Private Set Intersection Existentiality (PSI-X), which reveals to one party only one bit of information beyond the input size, whether or not the intersection is empty. PSI-X is not covered here and an in-depth study of the equivalences amongst these three primitives is ongoing work.

Additionally, some applications motivate alterations to the basic PSI protocol beyond output modification. Specifically, we examine PSI protocols that operate over authorized input. As discussed in Section 1.3 this requires slight modifications to the standard models of security. Most importantly we must introduce a semi-trusted third party that operates only as a signing authority and not an active participate. i.e, the new protocol entity signs one or both parties inputs before protocol interaction, after which no further involvement is required.

Definition 1.3 (Authorized Private Set Intersection (APSI))**.** *A three party protocol between Server with input* $\mathcal{S} = \{s_1, \ldots, s_w\}$*; Client with input* $\mathcal{C} = \{c_1, \ldots, c_v\}$ *and* $\mathcal{C}_\sigma = \{\sigma_1, \ldots, \sigma_v\}$*, where* $\sigma_i = \mathsf{Sig}(sk_{\mathsf{CA}}, c_i)$*, for* $i = 1, \ldots, v$*; and authorization authority CA with input* sk_{CA}*. At the end of the protocol, Client learns:* $(\mathsf{ASI} \stackrel{def}{=} \mathcal{S} \cap \{c_i \mid \mathsf{Ver}(pk_{\mathsf{CA}}, \sigma_i, c_i) = 1\}, |\mathcal{S}|)$*, Server learns* $|\mathcal{C}|$ *and CA learns nothing. APSI securely implements:* $\mathcal{F}_{\mathsf{APSI}} : (\mathcal{S}, (\mathcal{C}, \mathcal{C}_\sigma), sk_{\mathsf{CA}}) \mapsto (|\mathcal{C}|, (\mathsf{ASI}, |\mathcal{S}|), \perp)$.

The above protocols can be realized in a number of ways. Primarily, solutions can be grouped into one of two competing schools of thought. First, there are solutions based on general frameworks for secure computation, such as garbled circuits or oblivious random access memory. These protocols provide flexibility for increased bandwidth and computational cost. Second, there are "specialized" solutions based on a plethora of number-theoretic assumptions. This dissertation focuses on protocols of the second kind due to their increased practicality, especially in resource constrained environments. A detailed study of the pros and cons of both approaches is left for future work. Specifically, our implementations use the instantiations of PSI-CA, and APSI from the literature as defined below. These imple-

mentations surpassed their generic-based competitors at the first annual iDASH Privacy & Security Workshop due to their increased efficiency.

In addition to implementing the above known variants of PSI this dissertation defines, realizes and implements several closely related novel variants. For the most part, these variants are based loosely on the above instantiations and are introduced in their relevant sections.

The specific existing protocols implemented here are defined below:

Private Set Intersection Cardinality. We implement the PSI-CA protocol of [DT11], secure in the semi-honest model under the Decisional Diffie-Hellman (DDH) assumption in the Random Oracle Model (ROM). It executes on common input two large primes p, q, such that $q|p-1$, and two cryptographic hash functions H and H'. We assume that p, q, H and H' are publicly available and that the same values are used in all instantiations of a specific application in order to support pre-computation. (Computation below is assumed mod p.)

Client picks R_c randomly from $\mathbb{Z}_q$ and then, for each $c_i \in \mathcal{C} = \{c_1, \ldots, c_v\}$, computes $a_i = H(c_i)^{R_c}$. Then it sends $\{a_i\}_{i=1}^{v}$ to *Server*. The latter picks R_s and randomly from $\mathbb{Z}_q$ and sends client $\{a'_i = (a_i)^{R_s}\}_{i=1}^{v}$ after being shuffled. Server also sends, for each $s_j \in \mathcal{S} = \{s_1, \ldots, s_w\}$, $ts_j = H'(H(s_j)^{R_s})$.

Finally, client outputs $|\mathcal{S} \cap \mathcal{C}|$ as:

$$|\{ts_1, \ldots, ts_w\} \cap \{H'({a'_1}^{1/R_c}), \ldots, H'({a'_v}^{1/R_c})\}|.$$

Authorized Private Set Intersection. We implement the semi-honest APSI protocol of [DCKT10], executed on common input an RSA modulo $n = pq$, an RSA encryption exponent e, two random group elements g, g' such that $\langle -1 \rangle \times \langle g \rangle \equiv \langle -1 \rangle \times \langle g' \rangle \equiv \mathbb{Z}_n$, and

two cryptographic hash functions H, H'. All these parameters are assumed to be fixed for all instantiations of the protocol and publicly available.

For each element c_i in its input, client receives from the certificate authority σ_i such that $\sigma_i^e = H(c_i)$. Then client picks a random value $R_{C:i} \leftarrow \mathcal{Z}_{n/2}$ and two random bits $b_i, \overline{b}_i$ and computes $M_i = (-1)_i^b \cdot \sigma_i \cdot g^{R_{C:i}}$, $N_i = (-1)^{\overline{b}_i} \cdot H(c_i) \cdot (g')^{R_{C:i}}$. Then it computes a zero-knowledge proof $\pi = ZK\{R_{C:i}, i = 1, \ldots, v \mid M_i^{2e}/N_i^2 = (g^e/g')^{2R_{C:i}}\}$. Client sends $\{M_i, N_i\}$ and π to server. If π verifies, server picks a random element $R_S \leftarrow \mathbb{Z}_{n/2}$ and computes $Z = g^{2eR_S}$. Then, for each element M_i from Client, it computes $M_i' = (M_i)^{2eR_S}$ and for each of its input elements s_j, it computes $T_{S:j} = H'(K_{S:j}, H(s_j), s_j)$. Finally, it sends back to client $Z, \{M_i'\}, \{T_{S:j}\}$ and π' where $\pi' = ZK\{R_S | Z = g^{2eR_S}, \forall i, M_i' = (M_i)^{2eR_S}\}$.

If π' verifies, client outputs $\{T_{S:j}\} \cap \{H'(M_i' \cdot Z^{-R_{C:i}}, H(c_i), c_i)\}$.

1.2 Secure Pattern Matching

In addition to Private Set Intersection we cover the closely related primitives of Secure Pattern Matching (SPM) and Secure Substring Matching (SSM). While all of the intricacies of SPM are covered in great detail in Chapter 4 we describe it here briefly. Informally, one party contains a text t and the other a pattern p, possibly containing single character wildcards (capable of matching any character). The protocol reveals to the pattern holder a single bit of information, if the pattern matched the text at any location.

Our definitions rely on the following notation. The $t[x : l]$ notation signifies the l length substring of t rooted at x. i.e $(t_x, \ldots, t_{x+l})$. The wildcard character"w=*" has the property that $w = x$ evaluates to true for any x. Finally, two strings $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_m)$ are considered equal if and only if for all$i < m$ $x_i = y_i$.

Definition 1.4 (Secure Pattern Matching (SPM)). *A protocol between Client with input* $(p = p_1, ..., p_m)$*, and Server with input* $(t = t_1, ..., t_n)$ *in which Client learns* n *and if* $t[s : m] = p$ *for any* $s < n$ *and*[1] *Server learns* m*. Any* p_i *can be the wildcard character* w*,* t_i *must be letters from the alphabet. SSM securely implements:*

$$F_{SSM}(p, t) \rightarrow (m, (b, n)), \textit{where } b = \begin{cases} 1 & \textit{iff } \exists ss.t.t[s : m] = p \\ 0 & \textit{otherwise} \end{cases}$$

Secure Substring Matching (SSM) is SPM with the modification that the pattern must match exactly one location. Occasionally this is extended to a set of locations. SPM can be built by O(n) innvocations of any SSM protocol.

Definition 1.5 (Secure Substring Matching (SSM)). *A protocol between Client with input* $(p = p_1, ..., p_m, s)$*, and Server with input* $(t = t_1, ..., t_n)$ *in which Client learns* n *and if* $t[s : m] = p$ *for* $s < n$ *and Server learns* m*. Any* p_i *can be the wildcard character* w *or letter from the alphabet* Σ*,* t_i *must be letters from the alphabet. SSM securely implements:*

$$F_{SSM}((p, s), t) \rightarrow (m, (b, n)), \textit{where } b = \begin{cases} 1 & \textit{iff } \exists t[s : m] = p \\ 0 & \textit{otherwise} \end{cases}$$

We make two concrete contributions to the existing SPM literature. First, the design and analysis of a size hiding SSM scheme detailed in Chapter 4. Second, the design, analysis, and implementation of a limited SPM scheme with[2] detailed in Chapter 6. The latter is of particular interest as the implementation goes beyond a simple prototype. It is capable of processing complex pattern queries over databases with hundreds of millions of rows in real time. This is the most efficient protocol covered in this work; however, the additional efficiency comes at the price of decreased privacy. Unlike the majority of the other protocols

[1] The $t[x : l]$ notation signifies the l length substring of t rooted at x. i.e $(t_x, \ldots, t_{x+l})$

[2] Only some pattern characters can be wildcards

herein the solution requires an active semi-trusted third party capable of learning a function of access patterns over the data (called leakage).

In addition to pattern and substring matching a few applications are concerned with the related and well studied concept of hamming distance defined as follows.

Definition 1.6 (Secure Hamming Distance (SHD))**.** *: A protocol between Server with input string* $\mathtt{S}$*, and Client with input string* $\mathtt{C}$ *such that* $|\mathtt{S}| = |\mathtt{C}|$ *At the end of the protocol, Client learns* $\mathsf{HD}(\mathtt{S}, \mathtt{C})$ *(where* HD *denotes Hamming Distance, i.e., the number of positions at which the corresponding symbols are different) and Server learns nothing. SHD securely implements:* $\mathcal{F}_{\mathsf{SHD}} : (\mathtt{S}, \mathtt{C}) \mapsto (\perp, \mathsf{HD}(\mathtt{S}, \mathtt{C}))$.

We implement the following trivial protocol realizing SHD.

Secure Hamming distance (SHD). To obtain the SHD of two equal-length string in the semi-honest model, we can use any additively homomorphic encryption scheme, such as, Paillier [Pai99] or additive ElGamal variant [ElG85].

Client generates keypair (pk, sk) and, given string $\mathtt{C} = c_1 || \ldots || c_n$, computes $\{a_i = E_{pk}(-c_i)\}_{i=1}^n$. Similarly, *Server* computes $\{b_i = E_{pk}(s_i)\}_{i=1}^n$. Client sends $\{a_i\}_{i=1}^n$ to *Server*, which computes $d_i = b_i * a_i$, such that d_i is the encryption of $(s_i - c_i)$. Next, *Server* picks n random values, $\{r_i\}_{i=1}^n$, and returns $\{e_i = {d_i}^{r_i}\}_{i=1}^n$ to *Client*, after shuffling them. Finally, *Client* sets $z_i = 1$ if $D_{sk}(e_i) = 0$, and $z_i \neq 0$ otherwise, and computes $\mathsf{HD}(\mathtt{S}, \mathtt{C}) = \sum_{i=1}^n (z_i)$.

Our implementation uses the additive ElGamal encryption scheme, secure under the Decisional Diffie-Hellman (DDH) assumption in the standard model. Let p, q (s.t., $q|p-1$), and a generator g of a subgroup of $\mathbb{Z}_p^*$ of order q be public parameters. Let $x \leftarrow \mathbb{Z}_q$ be the private key sk and $y = g^x \mod p$ the public key pk. (pk is transferred to server as the first step of the interaction). Encryption of message $m \in \mathbb{Z}_q$ is $Enc_{pk}(m) = \langle c_1, c_2 \rangle = \langle g^r, h^r \cdot g^m \rangle$

for $r \leftarrow \mathbb{Z}_q$. Decryption is $Dec_{sk}(Enc_{pk}(m)) = c_2/{c_1}^x = g^m$, thus, one can efficiently test whether $Enc_{pk}(m)$ is an encryption of 0.

1.3 Highlighted Security Properties

Our protocols operate in several varied security models. Primarily we prove our schemes in the standard secure computation syntactic framework. In several instances we deviate from what is done typically in the literature. Specifically, we create protocols that offer bi-lateral security, both parties learn the output result; authorization, some or all inputs have been authorized by a third party; and/or size-hiding, input size is hidden from one party. We detail the basic adversary model below, followed by a more in-depth discussion of some of the more interesting deviations.

Adversarial Model. Unless otherwise noted we use standard security models for secure two-party computation, which assume adversaries to be either semi-honest or malicious. Hereafter, the term *adversary* refers to protocol participants, since actions of outside adversaries can be mitigated via standard network security techniques. Following [Gol04], protocols secure in the presence of *semi-honest adversaries* assume that parties 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 party might (passively) attempt to infer additional information about other party's input whereas security in the presence of *malicious parties* allows arbitrary deviations from the protocol. As this thesis focuses on practical applications of privacy preserving protocols, security arguments are often made with respect to *semi-honest* participants; however, efficient extensions to malicious participant security have already been developed for many of our cryptographic building blocks, and extensions to malicious are presented where applicable.

Briefly, the standard secure computation syntactic framework of [Gol04] in the semi-honest model corresponds to considering an ideal implementation where a trusted third party (TTP) receives the inputs of both parties and outputs the result of the defined function. Protocols are secure if in the real implementation of the protocol (without a TTP), each party does not learn more information than in the ideal implementation.

Nonetheless, semi-honest security definitions are not always sufficient for our needs. They typically don't capture the concept of *unlinkability* that refers to the impossibility for a party to learn whether any two protocol executions are related, i.e., executed by the other party on the same input. Additionally, they aren't suited to handle authorized input particularly in the case of bilateral or *two-way* protocols in which both parties operate over input that has been independently validated as authentic and each expects a result.

To this end we modify the standard definitions where applicable.

Bilateral Security. In the literature it is often mentioned that two simultaneous instances of a one-way (one party learns the result) secure computation protocol can lead to a two-way (both parties learn the result) secure computation protocol. We find for practical PSI this is not the case. Typical PSI protocols offer no mechanism to prevent an adversarial party from entering differing inputs into both executions. While a theoretical HbC adversary can be easily instructed to use the same input, when building for practical applications the power of the HbC assumption quickly falls apart. This problem is compounded when considering parties instructed to operate over authentic input. This issue is studied in greater detail in Chapter 3.

Size Hiding. The majority of secure computation protocols in the literature do not consider the size of parties input at all. For set intersection and pattern matching problems this information can be extremely sensitive depending on the application. Recent research has discovered that hiding size for both generic and specialized protocols is exceedingly difficult

and costly. Further, hiding the input size of both parties remains an open problem, and in some cases it is impossible. In line with this, our size-hiding protocols focus on hiding the size of only one participant. As covered in Chapter 5, size-hiding is of particular interest to genomic applications where, due to the small alphabet size, a pattern's size may be enough to identify the pattern in its entirety.

Traditional definitions and proofs of both PSI and SPM omit the input size from their descriptions. However, due to its importance to this work, our definitions typically include this information. Note, that while they appear slightly altered these definitions are in-line with the standard definitions; for example, the work of [FNP04]. That said, on occasion, we too will leave out these sizes where they are not relevant in order to ease exposition.

As others have discovered one of the most basic ways to hide size is by padding input elements to some fixed maximum. Such a solution, however, is both inelegant and often impractical. Especially when considering large inputs, as is the case in genomics. As such this solution, while feasible, is not covered in great detail.

Additionally, hiding size can have negative consequences. A party whose size is hidden can have an input as long as they wish. This is particularly important for set intersection where inputs may be drawn from a small domain. While not part of this work, we derive a solution dubbed Bounded Size Hiding Private Set intersection that addresses this issue by allowing a public cap to the size of any input[BFT16].

1.4 Practical Applications

The research presented in this work can be used in many applications. In this section, we briefly detail some of the most prominent motivators. As we will show, our implementations offer solutions to these problems that are practical today.

1.4.1 Genomics

As fast and accurate sequencing of human genomes becomes affordable, it is expected that individuals will soon be able to carry around copies of their sequenced DNA, using it for medical, identification, and social purposes. This will undoubtedly prompt a wide range of new and interesting genomic applications. However, the very same progress raises some worrisome privacy issues, since a genome represents a treasure trove of highly personal and sensitive information. Genomes are especially sensitive in comparison to other forms of data and require special precautions. Once information derived from ones genome is leaked, it cannot be revoked. Further, its efficacy lifetime may be nearly indefinite as genomic information is relevant beyond the affected individual.

PSI protocols and their variants allow us to keep this new and exciting frontier at significantly reduced risk. Chapters 2 and 5 give an analysis of the privacy challenges raised by the advent of ubiquitous genomic computing and how PSI and SPM can help.

1.4.1.1 Genomics Primer

Below is a detailed description of the genomic terms used in this work.

Genomes carry hereditary information needed to build and maintain an organism. Aside from certain kinds of viruses, genomes are encoded in double-stranded DeoxyriboNucleic Acid (DNA) molecules, i.e., two long polymer chains of four units called nucleotides. A nucleotide is represented by one of the four letters: A, C, G, and T. A human genome consists of around 3.2 billion nucleotides.

Whole Genome Sequencing (WGS) is the process of determining the complete and exact DNA sequence of an organism's genome. Today, sequencing techniques extract, from a DNA

sample (e.g., saliva, hair, nails blood and skin flakes), short DNA reads with hundreds of nucleotides, that are then analyzed and aligned to a so-called *reference genome*. This allows progressive reconstruction of the whole genome. Data produced by sequencing machines is usually in the form of a set of aligned strings, with associated accuracy scores. Thus, in order to represent a genome as input to SPH-PSM, we need to convert it to a single-string representation where each character in the string corresponds to the letter in the sequenced genome at the same offset.

Indels are occasional, naturally occurring insertions or deletions in genomes. A deletion happens when one or more base pairs are removed from a DNA segment. Analogously, an insertion represents a mutation where one or more nucleotides are inserted in a particular DNA fragment. Insertions are rare in human genomes and are not relevant for the majority of tests considered (see paragraph on SNPs below); thus we typically ignore them. Further, sequencing involves alignment to a reference genome and indels are always detectable. To contend with deletions we can introduce a special symbol '$-$' in lieu of a deleted letter at a specific location in the genome. This new symbol should be treated as any other base pair, and could potentially exist within a search pattern.

Single Nucleotide Polymorphisms (SNPs) are the most common form of DNA variation occurring when a single nucleotide (A, C, G, or T) in the genome differs between members of the same species or paired chromosomes of an individual [S$^+$09]. The average SNP frequency in the human genome is approximately 1 per 1,000 nucleotide pair. (See [Nat16] for a complete collection of all known SNPs). SNP variations are often associated with how individuals develop diseases and respond to pathogens, chemicals, drugs, vaccines, and other agents, and constitute the main focus of *personalized medicine* testing [Car08].

Restriction Fragment Length Polymorphisms (RFLPs) refers to a difference between samples of homologous DNA molecules that come from differing locations of restriction en-

zyme sites, and to a related laboratory technique by which these segments can be illustrated. In RFLP analysis, the DNA sample is broken into pieces (digested) by restriction enzymes and the resulting restriction fragments are separated according to their lengths by gel electrophoresis. Thus, in short, RFLP provides information about the length (and not the composition) of the DNA sub-sequences occurring between known sub-sequences that are recognized by particular enzymes. Although it is being progressively superseded by inexpensive DNA sequencing technologies, RFLP analysis was the first DNA profiling technique inexpensive enough to see widespread application and is still in use today. RFLP probes are frequently used in genome mapping and in variation analysis, such genotyping, forensics, paternity tests, hereditary disease diagnostics. (For more details, see [Nat11].)

Notation. In the rest of this work, we denote a digital copy of an individual's fully sequenced genome by $\mathcal{G} = \{(b_1||1), \ldots, (b_n||n)\}$, where $b_i \in \{\text{A, G, C, T, –}\}$, n is the genome length, and "||" denotes concatenation. The "–" symbol is needed to handle DNA mutations corresponding to *deletion*. In case of *insertion* mutation in the genome, e.g., an 'A' is added between positions x and $x + 1$, we add $(A||x||1)$. Similarly, if insertion involves multiple nucleotides. Since insertions are rare in human genomes (in the order of 0.1%), we typically do not consider them. Where relevant we discuss how to amend presented protocols to be insertion aware.

We use the $|str|$ to denote the length of string str and $|A|$ to denote the cardinality of set A.

1.4.2 Privacy Aware Consumer Applications

The recent decade has witnessed a rapid increase in popularity of mobile personal devices (notably, smartphones) t hat function as all-purpose personal communication portals. Concurrently, On-line Social Networks (OSNs) have continued their impressive proliferation. Meanwhile, the notion of "OSN privacy" remains elusive and even self-contradictory. Cen-

tralized nature of prominent OSNs is unlikely to change, which does not bode well for OSN users' privacy. However, some user privacy can be gained from making certain OSN functionality available **off-line**, such as discovering common contacts and other features, as well as establishing affinity-based connections. OSN providers stand to gain from this, since users could avail themselves of OSN functionality in scenarios where none currently exists, e.g., whenever Internet connectivity is unavailable, expensive or insufficient. At the same time, OSN users benefit from increased privacy because off-line interactions can be made opaque to OSN providers.

Chapter 3 explores off-line private proximity-based use of OSNs. Although our approach is quite general, the proposed system (called *UnLinked*) is grafted atop a specific and popular OSN – *LinkedIn*. The chapter describes and evaluates a practical prototype that allows physically proximate LinkedIn users to commit to a connection if they have a mutually acceptable number of common connections.

This work gives evidence that this and other consumer applications running on resource constrained devices are capable of utilizing PSI and its variants today. While the protocols are not inexpensive, they operate fast enough in most scenarios to be realized with little to no delay to the consumer.

1.4.3 Inter-Organization Information Sharing

Over the past decade, privacy issues have risen regarding the desire to share data between local law enforcement, state and local governments, first responders, and international governments. A recent example of such discord was the European Court of Justice ruling in May 2006 that ordered the cessation of passenger name record data-sharing with the U.S. Though the data required was relatively innocuous, it took two years to implement a new agreement. Further, privacy advocates within Europe remain who oppose the sharing of data without

stronger data privacy safeguards [KE09]. Even more difficult is sharing sensitive intelligence or law enforcement data among the tens of thousands of state, local, and tribal governments just within the United States alone. Balancing security concerns with information sharing remains a top priority for several intelligence and law enforcement agencies. Solutions must simultaneously provide strong protection for privacy and security while enabling access to real time intelligence and law enforcement databases. While many secure computation protocols can aid in this effort, secure pattern matching and private set intersection are especially useful. These tools can enable secure evaluation of an expressive set of queries to search such data including: exact matching, substring and approximate matching, and range queries for numerical data.

Many of the ideas and protocols in this work give partial solutions to these issue. That said, solutions provided in Chapter 6 are in many cases deployable today.

Chapter 2

Privacy Preserving Genomic Testing on Smartphones

This chapter addresses the problem of using Private Set Intersection to build efficient privacy preserving genomic testing on smartphones. It is based on my publication *Genodroid: are privacy-preserving genomic tests ready for prime time?* published in the Workshop on Privacy in the Electronic Society written along with Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik [DCFGT12]. While the presentation has changed no extenstions have been made.

During the last several decades, the scientific community made significant efforts to improve accuracy, and reduce the cost of, Full Genome Sequencing (FGS), making prices drop significantly faster than Moore's law would otherwise predict [NHG12, Sin12]. (See, for instance, the $3B, 13-year Human Genome Project [Int01] and [Kai08, Siv08].)

A genome represents the entirety of a specific organism's biological information. The availability of *fully sequenced* – and not only human – genomes naturally opens up new and exciting frontiers in numerous fields, including bioinformatics, genomics, genetics, and med-

icine. In particular, the vision of personalized medicine has been one of the driving forces behind FGS research. Its goal is a set of genomic tests that assess individuals' risk for major diseases such as diabetes and cancer as well as targeted screening and preemptive intervention [Cas10]. Indeed, genetic information already guides doctors toward accurate diagnosis and treatment. However, while some diseases (e.g., Huntington's) are caused by mutations in a single gene and are easily tested *in vitro*, the risk of developing other diseases depends on multiple genes which makes them difficult to identify. Low-cost genetic sequencing provides researchers with much more genomic information, and enables them to identify new genetic variations as well as run more complicated tests.

Full genome sequencing also facilitates other new applications, such as as paternity, ancestry and genetic compatibility testing. Although less critical than personalized medicine, these more "social" applications are no less exciting partly because of their (expected) broader appeal. Current lab-based, physical versions of these tests are both time-consuming and privacy-invasive. Performing them computationally makes them much more enticing and accessible.

More generally, we believe that in not-too-distant future, numerous genomic tests and operations will no longer be performed *in vitro* but *in silico*, i.e., using digitized genomes and specialized computational techniques [Hof07], possibly without the involvement of third-party testing facilities.

Despite numerous benefits of low-cost FGS, a number of serious ethical and privacy concerns have emerged [Col99, CM01, TBHB11]. Besides uniquely identifying its owner, a fully sequenced human genome contains information about one's ethnic heritage, phenotypic traits, and predisposition to numerous diseases and conditions, including mental disorders [FSC11, Can07, F$^+$09]. A virtual *treasure trove* of frighteningly personal and sensitive information is contained in one's genome. Traditional approaches to health care privacy, such as de-identification or aggregation [Mal05, H$^+$08] are not helpful in this context since the

genome is the ultimate identifier [MS00, MS01]. A recent study [SWH12] shows that a person's DNA could even be inferred from RNA data (often published in research repositories) even though it was previously assumed not to yield any information about its owner.

Consequently, in order for computational genetic tests on fully sequenced genomes to become accepted and commonplace, efficient and privacy-preserving versions of such tests need to be developed. This poses a number of challenges:

1. ***Privacy***: Given its extreme sensitivity, an individual should ideally never disclose personal genomic information. However, one should be able to allow others (e.g., individuals, doctors, or researchers) to run specific genetic tests that yield nothing beyond their intended results.

2. ***Accuracy***: Computational genomic tests should guarantee accuracy and reliability comparable to current (and widely accepted) lab-based *in-vitro* equivalents. For example, a software implementation of the paternity test on fully sequenced genomes should offer at least the same confidence as its *in-vitro* counterpart, currently admissible in a court of law.

3. ***Efficiency***: Computational genomic tests should incur minimal storage, communication, and computational costs, while satisfying privacy requirements associated with a given test type.

4. ***Portability and Accessibility***: Since a genome is arguably the most sensitive type of personal information, how and where should a user's genome (that contains about $3 \cdot 10^9$ letters) be stored? In the cloud? On a home PC? In a physician's office? On a smartphone? At the health insurance site? A closely related issue is: how should genomes be accessed?

5. ***Usability***: Computational genomic tests should be usable by, and meaningful to, regular non-tech-savvy users. This translates into non-trivial questions, such as: how

much understanding should be expected from a user running a test? What information (and at what level of granularity) should be presented to the user as part of a test and as its outcome?

Although widespread and affordable availability of fully sequenced human genomes makes it increasingly appealing to perform computational genetic tests, it also raises concerns in terms of simultaneously guaranteeing security, privacy and efficiency. The security research community has been attuned to the emergence of full genome sequencing and a few specialized privacy-preserving cryptographic techniques have been proposed in recent literature. However, to the best of our knowledge, practicality and usability of such techniques have not been assessed thus far. This is the main goal of this chapter.

By carefully designing privacy-preserving mechanisms that emulate *in-vitro*, highly accurate tests, our work demonstrates that secure computational genomic tests are viable today. We present a framework and an implemented toolkit, called *GenoDroid*. It incorporates several techniques offering efficient privacy-preserving genomic testing that meets most aforementioned challenges. In order to demonstrate ubiquity, GenoDroid runs on commodity Android smartphones (though it is not limited to this platform). We also conducted a pilot user study to explore usability and acceptability of proposed techniques.

We focus on the following tests:

- **RFLP- and SNP-based Paternity Tests** establish whether or not a male individual is the biological father of another individual, using genetic fingerprinting based on either Restriction Fragment Length Polymorphisms (RFLP) or Single-Nucleotide Polymorphism (SNP).
- **Ancestry and Genealogical Testing** allows individuals to trace their lineage by analyzing their genomic information. The scope of such tests is often quite hetero-

geneous. Ancestry testing is useful in a myriad of health-related applications (e.g., susceptibility to diseases common to certain populations). It is also increasingly used in social or recreational scenario, e.g., to map one own genetic heritage or find common ancestry.

- **Personalized Medicine (PM) Testing** provides a significant paradigm shift in health care, aiming at a more precise and powerful type of medicine [WH04], where diagnosis, treatment, and medication is tailored to the precise genetic makeup of the individual patient. For example, the US Food and Drug Administration (FDA) already recommends testing for mutations in the thiopurine S-methyltransferase (*tpmt*) gene, prior to prescribing 6-mercaptopurine and azathioprine – two drugs used for treating childhood leukemia and autoimmune diseases [Abb03].

Due to extreme sensitivity of human genomic material, for each considered test, we design and implement a privacy-preserving protocol that securely realizes the corresponding computation. Our protocols only yield the test results and do not disclose individuals' genomic information. Furthermore, if the the nature of the test involves sensitive information (e.g., it is a trade secret or is covered by a patent), the contents of the test are also concealed.

2.1 GenoDroid Framework

As discussed, full genome sequencing is revolutionizing diagnosis and treatment of certain diseases while producing new and more effective techniques for personalized medicine as well as ancestry and genealogy discovery. The next logical step is to transform paper-based research results into actual working computational tests available to individuals. As mentioned above, this poses challenges pertaining to usability, portability, accuracy, security, privacy and efficiency. To this end, our work focuses on the construction of efficient and privacy-

preserving techniques that allow individuals to perform genomic tests while disclosing only the required minimal information to other parties. Furthermore, we aim at ubiquitous availability of genomic tests, by designing protocols that run on current (off-the-shelf) Android smartphones.[1]

2.1.1 Smartphone Rationale

We chose to focus on the smartphone environment for several reasons, chief among them is the pervasive proliferation of smartphones into many spheres of everyday life [Can16] and their tendency to take over tasks previously relegated to desktop or laptop computers [ATT10]. Furthermore, the demand for smartphone use in health care applications is skyrocketing [PP12, SK10]. Modern smartphone's unparalleled portability makes it a true anytime-anywhere computing device and its highly personal nature (even laptops are often shared) motivates using it to store private information, such as cryptographic keys, PINs/passwords as well as one's genome.

Furthermore, computational power and storage capacity of today's smartphone are comparable to those of a laptop from a few years ago. Also, smartphone vendors and mobile OS developers (e.g., Apple, Google, RIM, and Microsoft) provide programming environments that facilitate quick and efficient implementation of complex applications. From the user's perspective, smartphones are relatively easy to use and are customarily carried almost everywhere. Thus, we believe that smartphones represent a viable and an appealing platform for performing personal genomic computations.

Clearly, the smartphone is not the only choice. A genome could also be stored at a physician's office. However, an individual may visit many types of medical specialists and/or change physicians. Secure storage, replication (e.g., if a specialist needs a copy) and migration

[1] Source code for all GenoDroid applications and framework components is available at `http://sprout.ics.uci.edu/projects/privacy-dna`.

(e.g, from one physician to another) are not trivial issues. Moreover, with health care costs already very high, the public would be unhappy to bear the costs of additional insurance that doctors would incur in order to protect their patients' DNA. Another option is to store and process genomes on a more powerful computing device, e.g., a desktop or a laptop. In both cases, portability is a major issue since a desktop is generally stationary, whereas a laptop, though portable, is much more burdensome to carry than a smartphone. This would rule out or limit social and recreational types of genomic tests (e.g., paternity or genetic compatibility). As mentioned above, desktops and laptops can be shared by multiple users, thus, making them more vulnerable to attacks.

Alternatively, genomes could be stored in the increasingly omni-present and (hopefully) benevolent cloud. Like a smartphone, the cloud allows anytime-anywhere access and offers computation services, in addition to storage. However, the cloud also requires reliable and pervasive Internet connectivity for its clients. A cloud vendor is also subject to unpredictable service outages and, of course, DoS/DDoS attacks. Moreover, cloud vendor privacy breaches can and should be expected; therefore, storing highly sensitive personal information in the cloud is perhaps not advisable. Even if a genome is stored in its encrypted form, unless and until fully homomorphic encryption becomes practical, complex computation on encrypted genomes stays out of reach. Naturally, in the course of a genomic test, an encrypted genome can be communicated from the cloud to the user's smartphone and the latter could perform the necessary computation. The main problem with this scenario is its cost; recall that a genome includes about $3 \cdot 10^9$ symbols. Finally, there is an issue of encryption longevity: a genome encrypted with, say, 128-bit (equivalent) key today is likely to remain secure for a few (20-25?) years. However, assuming that the cloud never "forgets" its hosted data, we need to wonder how secure would the same encryption key be 30 or 40 years from now.

2.1.2 Framework Structure

GenoDroid incorporates a number of building blocks for privacy-preserving genomic computations, e.g., decoding DNA strings produced by sequencing machines, privacy-preserving protocols, as well as auxiliary components, such as mutual authentication and device (smartphone) pairing.

One key feature of GenoDroid is its *extensibility*: although this chapter focuses on only three concrete genomic tests, our framework facilitates the development of other types of tests without re-implementing basic components from scratch. Future chapters use the cryptographic module of GenoDroid for rapid prototype development.

Overview. The framework currently supports two flavors of genomic tests: (i) both parties run on input of their respective entire genomes, e.g., to perform a paternity or ancestry test, or (ii) one party's input is an entire genome, while the other's – is a short sequence of *letter-position* pairs, e.g., a disease marker for personalized medicine tests. (Note that, in the latter case, letters do not have to be consecutive.)

GenoDroid integrates offline non-interactive pre-processing (e.g., on a desktop or a laptop), with online interactive computation on smartphones. Thus, computation occurs in two phases: (1) the entire genome is pre-processed, yielding a representation suitable for a given genomic test, and (2) the actual test is performed as a two-party protocol, where at least one party uses a smartphone. The pre-processing phase is particularly appealing since it separates software development from the knowledge of biological details that are not directly related to specific tests. For example, all genomic tests discussed in this chapter, as well as most others, require conversion of raw output produced by a sequencing machine to a "single-string representation".[2]

[2]Incidentally, to the best of our knowledge ours is the *first* concrete implementation of this functionality. In fact, standard genomic tools rely on a multi-string representation for human genomes, and all publicly available fully-sequenced genomes are encoded using multi-file formats (e.g., [10016]).

2.1.2.1 Pre-processing Components

Genomic data conversion. Independent of the specific test, we need to convert data produced by the sequencing machine in genetic laboratory – i.e., a set of aligned strings with the associated accuracy score – to a single-string representation, where each letter in the string corresponds to the letter in the sequenced genome at the same offset. GenoDroid can support most common formats currently used by sequencing labs, i.e., SAM, BAM, FastQ, and FastA. However, in our experiments, we use BAM-formatted files downloaded from the publicly available *DNA database* [10016]. This BAM format provides access to individual fragment reads, their alignment and accuracy scores, as reported by the sequencing equipment. For this particular format, our implementation uses the popular *BamTools* library [D. 16] to access raw (binary) data.

Test-dependent genome pre-processing. As discussed in the rest of the chapter, computational genomic tests often require an offline phase whereby the entire genome is scanned/processed to emulate *in-vitro* techniques. This is often needed to reduce the size of the input to the secure computation protocol that performs the test. GenoDroid includes a number of common DNA operations, e.g., digestion, probing, and sampling.

Cryptographic pre-processing. As mentioned earlier, privacy-preserving genomic tests in GenoDroid are based on two-party cryptographic protocols that yield nothing beyond intended test results. In many occasions, such protocols entail certain operations that can be pre-computed once offline. Offline costs can then be amortized over numerous (online) protocol runs. Such operations include key generation as well as protocol-specific pre-processing. In some cases, input to cryptographic pre-processing operations may depend on output of genome pre-processing (described above).

Each pre-processing component is assumed to be executed on a desktop or a laptop computer. (All information used in this phase must be securely erased immediately thereafter. Secure erasing is a well studied problem [Gut96], and is out of the scope of this dissertation.) We utilize all available computing cores and facilitate single-pass genome processing, i.e., whenever possible, operations are executed *concurrently*, so that information is read from disk only once. We also minimize memory usage, especially during pre-processing, and do not assume that, at any time, the entire genome is stored in RAM.

2.1.2.2 Smartphone Components

Once the pre-processing phase is completed, results are transferred to an Android smartphone. All smartphone-resident GenoDroid code is written in Java and executed in the Dalvik virtual machine [Goo16], a fast mobile-friendly JVM implementation that supports Just-in-Time compiling. As of mid-2012, high-end Android smartphones typically comes equipped with 1GB RAM and a dual- or quad-core ARM A9 processor running at 1.2-1.7GHz.

Secure Computation. GenoDroid implements several two-party cryptographic protocols, optimized for the Android platform: PSI-CA, SHD, and APSI. They represent the main building blocks for the genomic tests presented in this chapter. However, additional protocols can be easily integrated; in fact, GenoDroid already includes Private Set Intersection (PSI) from [DT10].

As mentioned above, we aim to minimize online (smartphone-bound), and maximize offline (desktop-bound), computation in all cryptographic protocols. However, in some cases, this can hinder protocol *unlinkability*. For example, in the SHD protocol we could minimize client's online work by pre-computing all public key encryptions. However, server would then learn whether client input changes over multiple interactions. One possible remedy

is to perform pre-computation for multiple interactions. This would put a strain on the smartphone's storage. However, some operations can be pre-computed, periodically, on the smartphone too, e.g., when it is idle and connected to an external power source. On the other hand, the argument for unlinkability is, in general, not particularly convincing in the context of personalized genomic tests since one's DNA stays (almost) the same throughout one's life and, even in a social setting, one is not likely to conduct, say, a paternity test with a random stranger.

Communication. GenoDroid supports multiple wireless communication technologies, selectable based on specific test requirements. It currently supports secure and authenticated communication over Bluetooth, Wi-Fi and cellular networks; this includes device discovery and secure device pairing. Depending on the underlying communication technology, we use different device pairing techniques. We use Bluetooth v2.1 (or higher), which offers Secure Simple Pairing (SSP) [Blu07] that allows two parties to bootstrap a public key authenticated channel. Over Wi-Fi, we perform both local (broadcast-based) and server-aided discovery. When using the cellular network, we also use server-aided discovery. This allows two parties, in two broadcast domains, to find a common rendezvous point. To do so, we implement a publicly available server, to which devices can advertise their presence using a human-readable ID and the fingerprint of their public key. As an alternative, the parties could identify themselves using X.509 certificates or anonymous credentials [CL01]: afterwards, the server reveals the counterpart's IP address and both parties can communicate directly. (If one or both parties are behind a NAT box, they would have to use the server to tunnel data.) Finally, remark that all tests presented in this chapter will be instantiated over Bluetooth.

Additional Components. GenoDroid also includes auxiliary components that implement storage management and user interface design. Since their development prompts no research issues, we do not discuss them in detail.

2.2 Paternity testing

We use the paternity test as one of the "measuring sticks" for assessing viability of privacy-preserving genetic computations. This might not seem like a natural choice: a paternity test is not a trifle but a highly personal matter, i.e., not something we could imagine doing in a social environment. It is also usually not performed upon a routine visit to a doctor's office. Current applications of this test are generally found in legal or law enforcement settings. However, since it is arguably the most common genetic test today, we use it as a *gateway* to other types of tests. Coincidentally, as discussed below, it also happens to be the least expensive. Thus, if the targeted smartphone platform cannot handle a privacy-preserving paternity test, it would also not be able to support more complicated test types.

A genetic paternity test determines whether two individuals have a father-child relationship. In this section, we use *paternity test* to denote a protocol involving two parties (Alice and Bob) using their respective genomes as input such that they learn the binary outcome. A *privacy-preserving* version of the paternity test involves disclosing minimum amount of genomic information.

Experts claim that one individual is the father of another if the Hamming distance between their genomes is below a well-defined threshold. Thus, the two individuals could run an SHD protocol to obtain a privacy-preserving paternity test. However, this would be relatively inefficient: even without privacy, computing the Hamming distance between two whole genomes would generate traffic in the order of 1GB and require the comparison of about 3 billion elements. Such a protocol would be prohibitively expensive on smartphones. Alternatively, since about 99.5% of the human genome is the same, the two parties could, in theory, compare only the remaining 0.5%. Unfortunately, there is not enough current understanding of the genome structure to pinpoint exactly *where* this 0.5% occurs.

Our solution reduces the size of the problem by drawing from optimizations used in *in-vitro* tests. Specifically, we simulate court-admissible *in-vitro* RFLP-based paternity tests. DNA is digested using a set of restriction enzymes and a small number of fragments, typically between 19 and 25, is selected using probes, defined by well-known markers [End89]. If two individuals are indeed father and child, then, with very high probability, the length of these fragments matches for at least a given number of fragments. This method is very reliable: with 25 fragments, its accuracy is estimated to be about 99.999% [End89, Lan89].

There are several advantages in performing genetic paternity test computationally, rather than *in vitro*. From the privacy prospective, participants do not need to disclose to a testing lab their identity or their entire genome, nor do they have to deliver swabs to a third-party facility and wait for the outcome. Instead, they can learn the test outcome immediately. Furthermore, as recently shown in [BBD+11], RFLP paternity testing can be simulated in computation and a corresponding privacy-preserving construction can be obtained if fragment lengths are compared privately, using *Private Set Intersection Cardinality* (PSI-CA).

2.2.1 An Optimized Implementation

We now present GenoDroid implementation of privacy-preserving paternity test. It includes two versions: the first, similar to [BBD+11], uses PSI-CA as the underlying cryptographic building block, and the second – SHD. Our implementation supports Bluetooth as the communication channel between interacting parties, to demonstrate feasibility of our approach to location-aware, bandwidth-constrained and easy-to-bootstrap settings. However, GenoDroid seamlessly lets us choose Wi-Fi or cellular networks.

While designing this application, our main objective is to reduce online computational overhead – crucial to guarantee a positive user experience – through protocol optimization and maximal pre-processing.

Pre-processing. We emulate RFLP-based enzyme digestion and marker-based fragment selection, in a single pass. We design an algorithm that compares a genome to all markers in parallel. (According to genomics experts, each marker appears at most once in the entire genome, thus, as soon as a match is found, the corresponding marker is removed from the set of available ones.) As a result, a set of n (in practice, 25) elements (m_i, ℓ_i) is produced, where m_i is the i-th marker and ℓ_i is the length of the corresponding fragment. This set constitutes the input to PSI-CA/SHD protocol run on the smartphones (see below).

Furthermore, pre-processing also performs offline operations related to PSI-CA or SHD protocols. However, as discussed in Section 2.1.2.1, if different runs of the test must be unlinkable, we can either: (i) perform pre-computation multiple times and transfer the corresponding output to the smartphone, or (ii) let the smartphone periodically perform pre-computation when idle and connected to a power source.

Smartphone Interaction. Using RFLP-based techniques, we reduce privacy-preserving paternity testing to *privately comparing how many of the n fragments have the same length* across the two individuals. That is, after independently applying the digestion/probing algorithm, parties learn how many fragments have the same length, and nothing else. Even if the test result is negative, this does not reveal any sensitive information. Hence, security of this construction only depends on that of the cryptographic protocol used for private comparison.

Furthermore, since input to this protocol is very small (only 25 "lengths") it can be executed efficiently. Nonetheless, we develop an optimized implementation of both PSI-CA and SHD using pipelined communication, i.e., both parties start transmitting processed data as soon as it is available, without waiting for the entire computation to finish. While one could choose any PSI-CA/SHD instantiation, our implementation (and experimental eval-

uation) employs the protocol from [DT11] (for PSI-CA) and the one based on additively homomorphic encryption (for SHD).

2.2.2 Performance Evaluation

We now evaluate GenoDroid's implementation of privacy-preserving paternity test. Let *Client* denote the party that successfully completes Bluetooth discovery and initiates the connection, and *Server* – the other entity. This also corresponds to the *Client-Server* nomenclature in PSI-CA and SHD protocols that we use. Recall that only *Client* obtains the test result from the interaction, however, in our implementation, it communicates it to *Server* over the secure channel. (This assumption does not violate our security model, since we assume that both parties are semi-honest.)

Genomes used in our experiments are downloaded from the 1,000 Genomes Project [10016], and pre-processed with enzymes from [R. 16].

Our measurements, presented in Table 2.1, are performed on two Nexus Galaxy phones, running Android 4.0, with a 1.2GHz TI OMAP 4460 ARM Cortex-A9 dual-core CPU, and communicating over an encrypted Bluetooth channel (as discussed in Section 2.1.2.2). In our experiments, we used 25 markers (corresponding to 99.999% accuracy) and tested both PSI-CA and SHD variants. Specifically, we measured:

1. ***Offline Time*** – time for both *Server* and *Client* to perform pre-computation pertaining to PSI-CA/SHD, on the smartphone. (Only if unlinkability is desired.)

2. ***Online Time*** – interval between the time *Client* starts communicating with *Server* and parties outputting the final result.

Also, we compare our optimized PSI-CA implementation with its non-optimized counterpart (without pre-processing and pipelining). Run-times are averaged over 1,000 trials to minimize measurement variations. Bandwidth measurements include all information transmitted by both *Client* and *Server*.

Our tests show that the optimized PSI-CA [DT11] yields the best results. Additively homomorphic encryption-based SHD is about 1.5 times slower than PSI-CA in the online phase. Our tests also show that pipelining and pre-computation significantly enhance user experience, since they allow the online protocol run-time to be over 3 times faster.

From the security point of view, the two instantiations – while both secure under the Decisional Diffie-Hellman (DDH) assumption – rely on different models: the SHD construct is secure in the standard model, whereas, selected PSI-CA is instantiated in the Random Oracle Model (ROM).

2.3 Genetic Ancestry Testing

Genetic ancestry testing allows individuals to trace their lineage through the analysis of their genomic information. Increasing understanding and availability of fully sequenced genomes makes commensurably more effective to study how susceptibility to common diseases varies among individuals and populations [RJ10]. Also, Genome-Wide Association Studies (GWAS) [B$^+$07b] are gaining momentum as they study common genetic variants in different

	Offline		Online	
	Server	Client	Time	Bandwidth
Optimized PSI-CA	399 ms	383 ms	244 ms	14.1 KB
Optimized SHD	736 ms	507 ms	376 ms	31.5 KB
PSI-CA no pipelining no pre-computation	–	–	784 ms	14.1 KB

Table 2.1: Computation & Communication costs of GenoDroid paternity test. Online cost reflect wall-clock-based run-times.

individuals to see if any variant is associated with, e.g., a disease, and possibly correlate such disease to a given ancestry line.

Besides health-related application, ancestry testing is becoming more and more popular for personal and social purposes. For instance, an increasing number of people is interested in tracing biological relatives and researching genealogical records, and discovering their family histories. Others are searching for connections to ethnic groups or geographical locations. The business side of recreational genetics is growing very fast, with scores of companies already offering ancestry tests costing only a few hundred dollars [B$^+$07a].

Today's genetic ancestry tests analyze either: (1) mitochondrial DNA (mtDNA), based on sequencing of maternally-inherited DNA material, or (2) the Y-chromosome, based on genomic information transmitted from father to son [B$^+$07a]. In both cases, individual's genomic information is compared with that of a sample individual.[3] Several commercial entities (e.g., 23andMe [23a16]) maintain a collection of sample genomes from individuals belonging to different ethnic groups, and compare them against their customers' genomic information to understand how they relate to known ethnic groups.

Alternatively, ancestry testing can be performed on two individuals in order to determine their genetic relationship. In this case, individuals learn whether or not they are "related" or even their distance in their common genealogical tree. Additionally, since the Y-chromosome is passed essentially unchanged from father to son, tests based on such portion of DNA provide precise information about paternal lineage. Similarly, mtDNA-based tests offer insight into one's maternal lineage.

The availability of full genome sequencing will soon allow performing more efficient computational analogs of tests that are now conducted exclusively in labs. However, privacy issues must be taken into account by both users and testing companies. Users might be unwilling

[3]For improved efficiency, rather than comparing the whole mtDNA or Y-chromosome, labs usually compare only a few (e.g., 50) SNPs across the entire genome or focus on a subset of insertions.

to surrender their entire genomes, while companies might not wish to disclose their test details (which could represent proprietary information or trade secrets). Also, as in the case of paternity, computational genetic ancestry testing does not require parties to ship biological samples to a lab, thus, test results may be obtained significantly faster and without exposure of genetic material to third parties. Nonetheless, even without an external lab, ancestry testing between two individuals may pose privacy concerns, whenever parties may not be willing to mutually disclose their ancestry or their complete genomic information. To this end, we need a privacy-preserving protocol that allows two parties to determine the *extent of their genetic proximity* without revealing any additional information about their respective genomes.

A simple way to realize a privacy-preserving ancestry test is to allow two parties to compare their entire genomes in an oblivious manner. This way, they can learn their genetic proximity without leaking additional information. However, since genomes include around three billion letters, such computation would be rather inefficient, both in computation and communication overhead. Therefore, currently popular tests restrict the comparison to either mtDNA or Y-chromosome, obtaining a slightly less accurate, yet still meaningful, metric. Nonetheless, the size of the input to the privacy-preserving computation would still be relatively large. Specifically, there are about 16,000 nucleotides in mtDNA and 58 million in the Y-chromosome. This is unlikely to yield an efficient implementation on a smartphone. A better approach combines the use of mtDNA/Y-chromosome information with either the knowledge of a subset of SNPs suitable for the test (as currently performed by commercial labs) or, if this information is unavailable, selecting a random subset as described next.

2.3.1 Construction

Preliminaries. This section presents a protocol for secure genetic ancestry testing based on Jaccard similarity index [Jac01]. This measures the similarity of sets A and B, as $J(A,B) = |A \cap B|/|A \cup B|$. High values of the index suggest that two sets are very similar, whereas, low values indicate that A and B are almost disjoint.

To realize privacy-preserving computation of $J(A,B)$, we only need secure computation of $|A \cap B|$, since $J(A,B) = |A \cap B|/(|A| + |B| - |A \cap B|)$, and this can be done using PSI-CA.

However, when two parties compute the Jaccard index, with or without privacy, they incur computation and communication complexity (at least) linear in the size of their sets. Thus, if performed over a whole genome, this computation might be relatively expensive. In fact, for any new comparison, the Jaccard index must be computed from scratch – i.e., no information used to calculate $J(A,B)$ can be re-used for $J(A,C)$. As a result, an *approximation* of the Jaccard index is often preferred, as it can be obtained at a significantly lower cost, e.g., using so-called MinHash techniques [Bro97]. Informally, MinHash techniques extract a small representation $h_k(S)$ of a set S through deterministic (salted) sampling. This representation has a constant size $\mathcal{O}(k)$, i.e., independent from $|S|$, and can be used to compute an approximation of the Jaccard index, again as the ratio between the intersection and the union of the samples. The parameter k also defines the expected error with respect to the exact Jaccard index – it is bounded by $\mathcal{O}(1/\sqrt{k})$ [Bro97].

Observe that, while the computation of $h_k(S)$ also incurs communication and computation complexity linear in set sizes, it must be performed *only once* per set, for any number of comparisons. Thus, with MinHash techniques, evaluating the similarity of any two sets requires only a constant number of comparisons. Further, we can privately approximate the Jaccard index of two sets by executing PSI-CA on input MinHash samples (and not the entire sets).

Ancestry Testing. Using MinHash, we implement an efficient privacy-preserving genetic ancestry testing protocol that can be executed on smartphones. Our protocol leverages a pre-processing phase performed on a desktop or laptop computer. The pre-processing phase in our construction takes as input the set representation of an individual's mtDNA or Y-chromosome. It extracts a compact representation, using MinHash, and also performs the offline computation phase for the PSI-CA protocol of [DT11]. Finally, the two parties perform the (PSI-CA) online computation on their smartphones and obtain the test result, i.e., estimate *how similar* their genomes are, based on one of the following datasets: (1) a small selection of SNPs; (2) an entire Y-chromosome; (3) the whole mtDNA material; (4) all known SNPs (approximately 3 million).

2.3.2 Implementation Details

Our implementation realizes privacy-preserving genetic ancestry testing by privately (and probabilistically) comparing how many common SNPs, mtDNA or Y-chromosome base pairs two individuals share, using MinHash and PSI-CA from [DT11].

Pre-processing. Depending on whether the test is performed using SNPs, mtDNA or Y-chromosome, the offline phase of our protocol – performed on a desktop computer – involves slightly different computation. SNP-based testing requires iterating over the entire genome (about three billion base pairs) to extract all known SNPs. The output of this phase is composed of around 3 million elements, which we represent as $(b_i||loc_i)$ where b_i corresponds to the i-th SNP and loc_i to its position in the genome. The set $\{(b_1||loc_1), \ldots, (b_n||loc_n)\}$ is then used as input for the offline phase of the PSI-CA protocol in [DT11].

The pre-processing phase of both mtDNA- and Y-chromosome based tests consists in the offline phase of selected PSI-CA instantiation, executed on input the whole mtDNA (16,000

base pairs) or the Y-chromosome (58 million nucleotides), represented as in the SNP-based test.

Smartphone Interaction. Regardless of the dataset used for the test, the online computation involves the comparison of a subset of the parties' input, extracted using MinHash. Once they establish a connection, two parties negotiate a common salt, which is used by MinHash to extract k elements from their respective input. These k elements are then used as the input to the PSI-CA protocol.

2.3.3 Performance Evaluation

We evaluate GenoDroid's implementation of privacy-preserving ancestry test using the same setup as in Section 2.2.2. We measure the offline overhead as the time required to perform the PSI-CA pre-computation. Online cost is measured as the online part of PSI-CA. We also include time, bandwidth and pre-computation storage requirements of the protocol executed without the use of MinHash on the entire genome, mtDNA, Y-chromosome and all SNPs. We also perform our measurements on input 50 randomly-selected SNPs, to simulate the test performed by 23andMe (note that the actual SNPs used by 23andMe are not publicly known).

Measurements are presented in Table 2.2. We instantiate MinHash with $k = 10{,}000$, leading to an error of about 1% in the final result. Run times are averaged over 1,000 trials to minimize measurement variations.[4] Pre-computation storage requirements for MinHash are identical to that of performing the same test without MinHash – i.e., 8.5 MB for mtDNA, 366 MB for SNPs and 6.9 GB for Y-chromosome tests. Bandwidth measurements include all information transmitted by both parties. Similar to paternity test, *Client* denotes the party

[4]For tests longer than one day, running times have only been estimated by running tests on smaller inputs – this was possible as tested protocols incur linear complexities.

that successfully completes Bluetooth discovery and initiates the connection, and *Server* – the other entity, and this also corresponds to *Client* and *Server* nomenclature in PSI-CA [DT11].

	Offline			Online	
	Pre-comp. Size	Server	Client	Time	Bandwidth
Full Genome	358 GB	494 days	494 days	273 days	1544 GB
Y-chromosome	6.9 GB	9.5 days	9.5 days	5.3 days	29.9 GB
All SNPs	366 MB	11.9 hours	11.9 hours	6.6 hours	1.5 GB
mtDNA	8.5 MB	227.7 s	227.0 s	125.3 s	8.5 MB
MinHash (all tests)	–	–	–	220.6 s	5.2 MB
MinHash w/ pre-comp. (all tests)	depends on test	142.3 s	141.9 s	78.3 s	5.2 MB
50 SNPs (23andMe)	6.25KB	713 ms	711 ms	394 ms	27 KB

Table 2.2: Computation & Communication Costs of our Privacy-Preserving Ancestry Test.

Our experiments show that, without using MinHash, only mtDNA-based tests can be executed on smartphones in a reasonable amount of time and with acceptable communication overhead. Due to our choice of k, the use of MinHash improves the performance of all protocols. Specifically, it takes, 78 seconds with (and 220 seconds without) pre-computation, to execute privacy-preserving genetic ancestry testing between two parties equipped with Android smartphones, with *all* datasets. (In fact, k is a constant, thus, it is independent of size of the original sampled set).

The amount of space required to store precomputed values prompts an interesting tradeoff. While it is easy to justify the use of 8.5 MB of memory for storing the elements precomputed from mtDNA genetic material, users may prefer not to store 366 MB or 6.9 GB of data (in the case of SNPs and Y-chromosome respectively) and rather perform the whole PSI-CA protocol online, incurring an additional two minutes of computation.

It is interesting to observe that our simulation of the 23andMe tests shows that, with the appropriate knowledge, genetic ancestry testing can be performed in near real time on current smartphones.

2.4 Personalized Medicine

Recall that PM aims at identifying genomic information needed to accurately predict: (1) a susceptibility to a given disease, (2) the course of a disease, and (3) response to treatment. For example, before treating leukemia and other autoimmune diseases, physicians are required, by the US Food and Drug Administration (FDA), to test patients for certain mutations of the *tpmt* gene. This particular gene codes the enzyme responsible for metabolization of a number of drugs. Thus, sensitivity and toxicity response to these drugs varies according to *tmpt* mutations. In general, PM entails testing for numerous genetic markers, ranging from one to a few hundred mutations.

With growing availability and better understanding of genomic information, future healthcare will be tailored to the patient's DNA and genetic PM tests are soon likely to become commonplace. This prompts some concerns about the entities (e.g., hospitals, doctors, insurance carriers and labs) that could obtain genomic information to perform such tests. Besides obvious privacy issues due to bulk access to patients' genomes, there is a more subtle problem involving liability and long-term safety of genomic data. We alluded to it earlier, in Section 2.1.1. Entities handling genomic data need to demonstrate that it was treated appropriately and disposed of when no longer needed. Storing genomic information, even for a short time, creates potentially attractive targets for breaches and attacks.

One intuitive approach is to let the patient independently run specialized software over her genome and check for a match (or lack thereof) against a given drug's fingerprint. However, pharmaceuticals usually consider DNA fingerprints of their drugs to be trade secrets and thus are not expected to reveal them. At the same time, for every new drug, pharmaceuticals are required to obtain approval from some government agency, e.g., the Food and Drug Administration (FDA) in the United States. Therefore, an ideal scenario would be a privacy-preserving PM test, whereby: (1) the patient learns the outcome but not the drug's DNA

markers, (2) the patient is somehow assured that the drug has been authorized by the relevant government agency, and (3) neither the pharmaceutical nor any other party learns any information about the patient's genome.

The rest of this section presents a technique for privacy-preserving PM testing, implemented in GenoDroid. Similar to other GenoDroid tests, it runs in real-time and involves a patient with an Android smartphone and a tester, e.g., a medical lab or a physician's office. We begin by discussing a strawman construction that, albeit privacy-preserving, is not suitable for a smartphone platform and proceed to illustrate our practical cloud-aided approach.

2.4.1 Initial Construction

Similar to [BBD+11], our initial approach to privacy-preserving PM testing relies on APSI. We use the particular construction from [DCKT10] where the patient plays the role of *Server*, the tester – *Client*, and the FDA – mutually trusted CA. The APSI protocol from [DCKT10] is secure in the malicious model. However, we argue that malicious player security may be not needed for the PM setting that usually takes place in a medical lab or a physician's office. This is because there is generally some basic trust between a patient and a doctor (or a lab performing the test). Also, computational tests can be audited, e.g., if protocol transcripts are mandated to be kept. Furthermore, there could be severe consequences for malicious behavior, e.g., loss of medical license. Thus, we slightly modify the construction from [DCKT10] to only achieve semi-honest security.

Specifically, protocol executes on common input CA's RSA public key (N, e), a generator g of $\mathbb{QR}_N$, and two cryptographic hash functions H, H', modeled as random oracles. (All computation is performed mod N). For each element $c_i \in \mathcal{C} = \{c_1, \ldots, c_v\}$ in its input, *Client* obtains σ_i such that ${\sigma_i}^e = H(c_i)$, i.e., a CA-issued signature warranting authorization. The interaction starts with client sending $\{a_i = \sigma_i \cdot g^{R_{c:i}}\}_{i=1}^{v}$ (for $R_{c:i} \leftarrow \mathbb{Z}_{N/2}$) to server.

Then, *Server* selects $R_s \leftarrow \mathbb{Z}_{N/2}$ and returns $(Z = g^{2eR_s}, \{a'_i = {a_i}^{2eR_s}\}_{i=1}^{v})$. Also, for each element $s_j \in \mathcal{S} = \{s_1, \ldots, s_w\}$ in its input, *Server* sends $\{ts_j = H'(H(s_j)^{2R_s})\}_{j=1}^{w}$ to *Client*. Finally, *Client* computes $\{tc_i = H'(a'_i \cdot Z^{-R_{c:i}})\}_{i=1}^{v}$ and outputs intersection $\mathcal{S} \cap \mathcal{C} = \{c_i \in \mathcal{C} \mid tc_i \in \{ts_j\}_{j=1}^{w}\}$.

APSI-based privacy-preserving PM testing is as follows. The patient uses her smartphone that stores pre-processed genomic information as well as various cryptographic material. We assume that, well ahead of running the protocol and after positive clinical trials, the FDA granted authorization (*auth*) to the pharmaceutical for a specific DNA fingerprint (*fp*), corresponding to a genomic test. We denote $fp = \{(b_j^* || j)\}$, where each symbol b_j^* is expected to occur at position j of a fully sequenced genome, and $auth = \{H(b_j^* || j)^d \bmod N\}$, where N and d are FDA's RSA modulus and private key, respectively. The patient's private input is her entire genome. (This is unavoidable since the test fingerprint can occur anywhere.) The tester's input is $(fp, auth)$, as defined above.

Tester and patient engage in APSI protocol, as described above, and, at the end of the protocol, the tester learns whether the patient's genome matches *fp*, provided that *auth* is a valid authorization of *fp*. Furthermore, the tester learns nothing further about the patient's genome, and (2) the patient learns nothing about *fp* or *auth*.

We note that server-side (patient's) computation can be partitioned into *offline* and *online* phases. Specifically, $\{ts_j\}_{j=1}^{w}$ can be pre-computed once for any number of tests. Therefore, while offline computation is linear in the size of the genome, its online counterpart is linear in the number of tested *loci*. However, $\{ts_j\}_{j=1}^{w}$ values must still be transferred, implying that online communication complexity is linear in the genome size. This translates into several gigabytes and obviously hinders adoption on modern smartphones.

2.4.2 Cloud-Aided Variant

We now show how to modify the initial approach to make it amenable for smartphones. The main idea is to off-load some of the patient's (*Server*'s) computation to the cloud. However, we immediately acknowledge that involving the cloud triggers the risks discussed at the end of Section 2.1.1.

The patient needs to pre-process the genome and upload the result to a (semi-honest) cloud provider. Clearly, such pre-processing must include some form of encryption, to conceal the genome from the cloud. Moreover, the encrypted elements must be shuffled in order to (partially) hide access patterns from the cloud. When the test is conducted, the patient explicitly grants the tester (*Client*) access to cloud-resident genomic information, such that only the test result is learned. This operation must be efficient and should prevent the tester from colluding with the cloud provider and learning any additional genomic information.

Unlike paternity and ancestry testing, the pre-processing phase in this cloud-aided variant results in two output sets: one uploaded to the patient's smartphone, and the other – to the cloud provider. Despite the presence of the cloud provider, the actual protocol is still based on APSI from [DCKT10].

Pre-processing. Besides genomic pre-processing (e.g., format transformation), patient pre-computes, e.g., on a desktop, $\{ts_j = H'(H(b_j^*||j)^{2R_s})\}_{j=1}^{w}$ and uploads the result, after shuffling, to the cloud. This does not reveal any information about the genome. Then, public cryptographic parameters (N, e, g, H, H'), along with private random value (R_s), are uploaded to the patient's smartphone. (Note that there is no pre-computation on the tester side.)

Interaction. This phase transpires between a smartphone-equipped patient and a tester on a desktop or a laptop connected to the Internet. Together, they execute the online phase of

APSI, i.e., the tester sends the patient $\{a_i\}_{i=1}^{v}$ and receives $\{a'_i\}_{i=1}^{v}$. The tester then computes $\{tc_i\}_{i=1}^{v}$ and finds matching ts_j-s using the cloud provider. There are at least three ways for tester to do so:

1. Query the cloud using $\{tc_i\}_{i=1}^{v}$.
2. Download all $\{ts_j\}_{j=1}^{w}$ from the cloud.
3. Privately query the cloud using, e.g., Private Information Retrieval (PIR) [CGKS95].

GenoDroid currently implements (1) since it is the most efficient of the three. Of course, since the tester is running on a regular computer, we could choose option (2). However, even on a desktop with a wired Internet connection, downloading approximately 60GB of data is time-consuming for the tester and unscalable for the cloud provider.[5] As far as option (3), recent advances in, and optimization of, single-server PIR techniques have yielded promising results that might be efficient enough for the tester's desktop or laptop, especially if queried database is maintained in a cloud cluster [BPMO12]. However, since our emphasis in this chapter is on the smartphone platform, we leave this item for future work.

Threat Model and Security. We assume that each participant is semi-honest, i.e., interacts with the other two participants while following all protocol specifications. However, a participant might attempt to surreptitiously learn further information about others' inputs. The underlying APSI protocols, as mentioned earlier is secure in the semi-honest model, with two parties. In the presence of the cloud provider (the 3rd party), collusions should be considered. If a tester colludes with a cloud provider, the only danger is that the former might obtain the entire "encrypted" genome – i.e., does not learn any meaningful information. Whereas, if a patient colludes with a cloud provider, they could learn some information

[5]GenoDroid assumes 20-byte $H'()$, thus, since the genome contains approx. $w = 3 \cdot 10^9$ nucleotides, $\{ts_j\}_{i=1}^{w}$ values amount to $(20 \cdot 3 \cdot 10^9)$B = 60GB.

about the proprietary (to the pharmaceutical) DNA fingerprint *fp*. In particular, the colluding parties could learn which (letters, positions) of the patient's DNA are specified in *fp*. However, this can occur only for the portion of the fingerprint matching the patient's genome. The case of a tester colluding with a patient might seem uninteresting; however, even though a tester acts on behalf of a pharmaceutical, their collusion can result in leakage of portions of *fp*, similar to the previous case.

In summary, we envision multiple entities (including generic cloud providers, HMO-s and insurance companies) will offer genomic cloud services to the public. Our foray into privacy-preserving personalized medicine testing is just one example of what can be achieved by combining the power of cloud computing/storage with smartphones in genomic computation and calls for further research.

2.4.3 Performance Evaluation

We now evaluate GenoDroid's implementation of privacy preserving PM testing. To compare with previous work, we use the same genetic fingerprints as in [BBD+11], i.e., the ones describing mutations *hla-B*5701* and *tpmt*. The former is associated with extreme sensitivity to abacavir, an HIV drug, and its fingerprint is composed of 2 nucleotide positions; the latter is tested before prescribing 6-mercaptopurine to leukemia patients. The *tpmt* fingerprint contains 6 nucleotide positions.

Measurements presented in Table 2.3 were obtained using the same setup as in Section 2.2.2, i.e., we used two Android smartphones communicating over Bluetooth. As discussed in Section 2.4.2, the tester can use a desktop computer and parties could communicate over Wi-Fi. However, we use Bluetooth for consistency's sake and in order to provide conservative results: our measurements represent an upper bound for the cost in a real-world setting.

	Online		Bandwidth	
	Patient	Tester	Tester-Cloud	Tester-Patient
*hla-b*5701*	78 ms	141 ms	0.40 KB	0.40 KB
tpmt	187 ms	301 ms	1.77 KB	1.77 KB

Table 2.3: Computation & Communication costs of GenoDroid's cloud aided hla-b and tpmt tests. Online cost reports wall-clock running time.

Patient and tester interact via an APSI protocol where the patient acts as *Server*, and the tester – *Client*. Client does not perform any precomputation, and we assume that *Server* uploads its encrypted genome to the cloud ahead of time. Measurements reported in Table 2.3 reflect the online phase of the APSI protocol. The Cloud does not perform any cryptographic operation, and therefore, we do not consider any of its computation costs. Bandwidth is measured for tester's interaction with patient and with cloud separately (bandwidth measurements are independent of the transmission medium).

Our experiments show that both tests for *tpmt* and *hla-b* can be executed on smartphones in well under a second. Our implementation compares favorably with that of [BBD+11] in terms of both bandwidth – due to the use of cloud-aided computation – and offline computation. Although our online phase runs slower than experiments reported in [BBD+11], we emphasize that (1) our timing include not only the cryptographic blocks, but rather a whole implementation over networked devices; (2) the Android smartphones used in our tests are significantly slower than the desktop platform used in [BBD+11].

2.5 Usability Study

One of the goals of the GenoDroid framework is to guarantee portability and accessibility, to average non-tech-savvy users, of privacy-preserving genomic tests. Thus, usability is one of the key factors influencing its acceptance. To this end, we conducted a usability study to

obtain feedback on our prototype applications as well as to assess the sensitivity of subjects to privacy concerns related to their genomic information.[6]

The study was conducted on 16 subjects (8 male and 8 female), sampling a diversified population of students, researchers, and non-scientific personnel. 90% of subjects belong to the 25-to-40 age range. Applications were run over mock human genomes, with same length and comparable nucleotide distribution as real human genomes. Our test mock-up was implemented using two Android-equipped Samsung Galaxy Nexus phones and used Bluetooth as the wireless communication medium.

As discussed in Section 2.2, we use *paternity test* as one of the "measuring sticks" for assessing viability and usability of privacy-preserving genetic computations. Arguably, paternity test is the most common genetic test today and non-tech-savvy users would likely relate to it more than to any other test. The chain of events is as follows: since the pre-computing phase takes place on desktops, the app is pre-loaded with the result of the RFLP-driven digestion and probing. Once the app is launched, the user is prompted with the Android interface to carry out secure Bluetooth pairing with another device and establish an authenticated and encrypted channel. Upon successful connection establishment, both users see a *"Start Test"* button; the first user to click is prompted with a *"Waiting for other party"* message, and, after the counterpart also hits start, the test is initiated. Finally, users are displayed with the test result, i.e., *"Tested individuals are/are not father and child"*. (In our user studies, the result is negative.)

After the test, subjects were asked to fill out a questionnaire corresponding to Brooke's well-known 10-item System Usability Scale (SUS) [Bro96], where answers indicate the degree of agreement or disagreement with the corresponding statement on a 5-point scale. Subjects rated application's usability, on average, at 82/100 on the SUS scale. Such a high score

[6]Our study received the "Exempt Registration" status from UC Irvine's Institutional Review Board (IRB).

can be safely considered above industry average and confirms that the perceived usability of the software is high enough to be deployed in practice. This is not surprising since the application is straightforward to use, does not require any high-end technical skill, and test running time is extremely low (in fact, a delay of 250ms is barely noticeable by users, and provides a seamless experience [Nie97]).

We also asked subjects to answer a few questions about privacy concerns related to genomic information. Subjects were asked to indicate their agreement on a statement using a 5-point scale, where 1 corresponds to "strongly disagree" and 5 – to "strongly agree". A few interesting findings follow. Subjects were "concerned with potential privacy exposure of (their) genomic information" (average 4.21/5 agreement). Somewhat more surprising is that our test subjects were "concerned with privacy even if tests are beneficial to (their) health" (average 3.08 agreement), while they were "in favor of genetic tests if they do not invade (their) privacy" (4.81 average agreement). Note that subjects in our study are *not* privacy/security researchers.

Chapter 3

Private Proximity-based Off-line Social Network Interaction

This chapter addresses the problem of bi-lateral (two-way) private set intersection over authorized input and its use in providing trusted offline social network interactions on smartphones. It is based on my publication *UnLinked: Private Proximity-based Off-line OSN Interaction* published in the Workshop on Privacy in the Electronic Society written along with Ronald Petrlic and Gene Tsudik [FPT15]. While the presentation has changed no extentions have been made.

Building trust in previously unfamiliar people based on common factors – such as interests, backgrounds, friends, or co-workers – has been practiced by the human race since time immemorial and has been thoroughly studied as a subject. In the last decade, due to the popularity of on-line social networks (OSNs), the process of finding and connecting to other people (based on something in common) has become easier due to OSNs' integrated search functionality and ability to trawl through public profiles and friends lists of one's own friends.

At the same time, despite very broad appeal, OSNs have encountered certain limitations to their proliferation. One reason is the fundamental connectivity requirement, i.e., the "O" in OSN: in order to use an OSN, one must be connected to the Internet and logged into the OSN provider. This is not surprising since most OSNs – including Twitter, Facebook, LinkedIn, VK and RenRen – are centralized.[1] In other words, there is no option for disconnected OSN usage. Consider the following scenarios:

- Low bandwidth or intermittent Internet connectivity, e.g., due to lossy and/or error-prone wireless links.
- Expensive Internet connectivity, e.g., abroad, on trains, planes and cruise ships.
- Complete lack of Internet access, e.g., in planes, under water, under ground or in remote locations.

We believe that such scenarios are fairly common for OSN users who travel. and they share a common feature in that OSN access is difficult: too slow, too expensive or simply impossible. However, there is no fundamental reason why two nearby OSN users – who either have no OSN access or do not want to connect to the OSN, could not have some *limited* OSN functionality. This observation is the premise and one of the motivating factors for this chapter.

Our second motivating factor is privacy. In general, lack of privacy is not a fair complaint against OSNs. Most people join an OSN for social reasons and privacy is not their primary concern. Although privacy advocates often decry brazen collection, marketing, mining and selling of OSN-derived user information, expecting OSNs to behave in a privacy-friendly manner is unrealistic. On one hand, it seems reasonable to observe and retain behavior (i.e., actions) and locations of users connected to the OSN. On the other hand, if OSN users are

[1] Although decentralized OSNs exist, e.g., Diaspora [Dia13] and Safebook [CMS09], they have not managed to attract many users.

communicating off-line, i.e., without involving the OSN infrastructure, it is no longer clear whether the OSN ought to have access to user behavior and location.

We note that there are two types of off-line user (inter-)actions: (1) those that lead to direct consequences to the OSN, and (2) those that do not. For example, consider two OSN users: Alice and Bob, who interact *verbally and in-person* while being disconnected from the OSN. During the chat, they exchange information about their friends, work history and educational background. If they have nothing in common, they do not subsequently connect on the OSN. However, if they discover some common factors (e.g., some number of shared friends), they might decide to connect later. In the former case, the OSN clearly learns nothing about their encounter. In the latter, the OSN observes spontaneous establishment of their subsequent connection.

By analogy with the above example, suppose that Alice and Bob interact electronically while being off-line (with respect to the OSN) and their interaction leads to some impact on the OSN, e.g., they later connect or "friend" each other, thus changing their profiles. In this case, the OSN will rightfully learn about their prior off-line interaction. Otherwise, if Alice's and Bob's off-line activity does not lead to anything and there is no reason for the OSN to learn about their off-line interaction.

To summarize, this chapter is prompted by the need to support limited off-line interaction between nearby OSN users. We believe that supporting this type of interaction would be beneficial for OSN users, for two reasons: (1) they would engage in social networking in a wider range of settings, and (2) they would do so knowing that positive outcomes can lead to new OSN connections, while inconsequential activity remains private. Furthermore, off-line user interaction is advantageous to OSN providers, since it would extend the reach of social networking. At the same time, we recognize that not all privacy issues stem from the OSN itself. If users communicate directly with no OSN involvement, their mutual privacy is very

important in cases that do not lead to a later OSN connection. We make this one of the main design goals.

Another key goal is information authenticity. When two OSN users interact on-line (via an OSN provider), information in their profiles can not be changed arbitrarily. For example, friends in Facebook or connections in LinkedIn are not added gratuitously. In the context of off-line interaction, we need to make sure that OSN profile information exchanged as part of that interaction is authentic and corresponds to the appropriate users.

In this chapter, we design an architecture and a system, called *UnLinked*. The main idea is to combine users' social proximity with their physical proximity to privately discover common factors and later possibly establish OSN relationships. *UnLinked*supports private off-line discovery of nearby users with authentic common friends or connections, without direct user interaction. Although conceptually applicable to many current OSNs, *UnLinked* is grafted onto one specific OSN, *LinkedIn* aimed at professionals who use their profiles as a sort of an online CV.

This work makes several technical contributions in addition to the overall *UnLinked* system design. As part of *UnLinked*, we come up with an efficient *Authorized Two-Way Private Set Intersection (ATW-PSI)* protocol and demonstrate its security. Current protocols are one-way, which means that only one party learns the intersection of the two input sets. Furthermore, they only certify one party's inputs. In our ATW-PSI, both participants learn the intersection only if each has a valid authorization issued by a *trusted third party (TTP)*. This has been identified as a major problem with prior techniques, e.g., [DCMP13]. Moreover, in contrast to prior work on protocols with linear complexity [DCT09, JL10, NDCD+13], participants in our protocol can not transfer authorizations for individual set elements to others. We also present a new approach to the well-studied *friend of friends (FoF)* discovery problem. By using *LinkedIn* as a concrete OSN platform and developing

a fully functional prototype[23] of *UnLinked*, we show that ATW-PSI protocols are usable in realistic settings.

3.1 Design Goals

Before proceeding with the system design, we overview and motivate key desired features that are mainly derived from the above discussion:

D0 **Off-Line Interaction**: support for efficient communication among two physically proximate OSN peers.We restrict the number of peers to two since the ultimate outcome of a fruitful off-line interaction is a two-way OSN connection.

D1 **Peer Anonymity:** persistent user identifiers (names, user ids, email addresses) associated with OSN members must be kept confidential in off-line interaction, unless the two decide to connect later by jointly revealing their identities at the end of off-line interaction. We require anonymity despite user's apparent proximity, since, in many situations the peers may be physically close, yet still unaware of each other's precise location or other identifying information.

D2 **Profile Privacy wrt Peers**: ability to perform certain OSN profile operations (e.g., compare respective sets of friends/connections) with mutual privacy. In other words, information learned from such operations must be limited to what is common to both peers.

D3 **Interaction Privacy wrt OSN:** non-disclosure to the OSN of off-line peer interactions and their locations. This specifically applies to interactions that have no impact on OSN peers' profiles, i.e., no eventual connection establishment.

[2]The prototype Android app can be downloaded from https://db.tt/XQXE9pqF.

[3]Due to a recent change in the LinkedIn developer agreement, our latest prototype uses Twitter as an example OSN.

D4 **OSN Connection Spillover:** ability to later establish actual OSN connections, based on prior off-line interaction that resulted in mutual agreement to connect. That is, it should be possible for two peers to connect via the OSN at some point when they are on-line, if they have decided to do so as a result of sufficient degree of commonality among their profiles, e.g., at least k shared friends.

D5 **OSN Profile Authenticity and Owner Authentication:** authenticity (including timeliness) of OSN profile information as well as authentication of each peer as part of off-line interaction. This is needed to protect OSN members from impostors (non-members) as well as malicious members.

D6 **OSN-Independent Operation:** ability to operate along-side (or on top of) an existing OSN, i.e., no requirement to introduce changes within an OSN; also, no restriction against an OSN implementing proposed functionality.

D7 **OSN-Agnostic Design:** minimal reliance on OSN-specific features, i.e., applicability to the broad spectrum of OSNs.

D8 **Voluntary Participation:** OSN users must *opt in* to participate in off-line interaction.

Note that off-line interaction between users of *different* OSNs, while desirable in the long term, is not among our goals for now.

3.2 *UnLinked* System Design

Based on the goals outlined above, our system architecture (called *UnLinked*) includes two main software components:

1. *UnLinked* App (ULA): an application that runs on the OSN user's personal device, such as a smartphone or a laptop, that takes part in off-line interaction, and performs auxiliary tasks on-line. ULA primarily supports D0, D2, D3, described in Section 3.1 above.

2. *UnLinked* Server (ULS): a stand-alone server program that supports D1, D5, D6, D7 and D8.

There are several practical reasons for ULS to be stand-alone, as opposed to being a component of an OSN. First, it allows us to support desired off-line functionality without any involvement of – or permission from – any specific OSN provider (D6). Second, ULS can be expanded to support multiple OSNs (D7). Third, ULS can operate as a registration portal where OSN users can enroll to participate in off-line interaction (D8). Fourth, acting as a neutral and independent trusted third party (TTP), ULS can certify (authenticate) profile information of OSN users (D5). Last but not least, ULS serves as a sort of a *privacy buffer* between the OSNs and ULA users. Although it can be argued that, from the complexity perspective, it makes more sense to integrate ULS into the OSN itself, independent operation of ULS insulates the OSN from its natural lack of motivation to respect user privacy (D3). That said, most privacy guarantees still hold if ULS is integrated into the OSN.

3.2.1 OSN Requirements

Following D7, we need to minimize assumptions about the underlying OSN. The only requirement of *UnLinked* is that the OSN must offer a secure automated way to export member profiles in some well-defined format, i.e., something that can be parsed by ULS. This feature is supported, via REST APIs, by most major OSN providers, such as Twitter, Facebook, Google+ and *LinkedIn*.

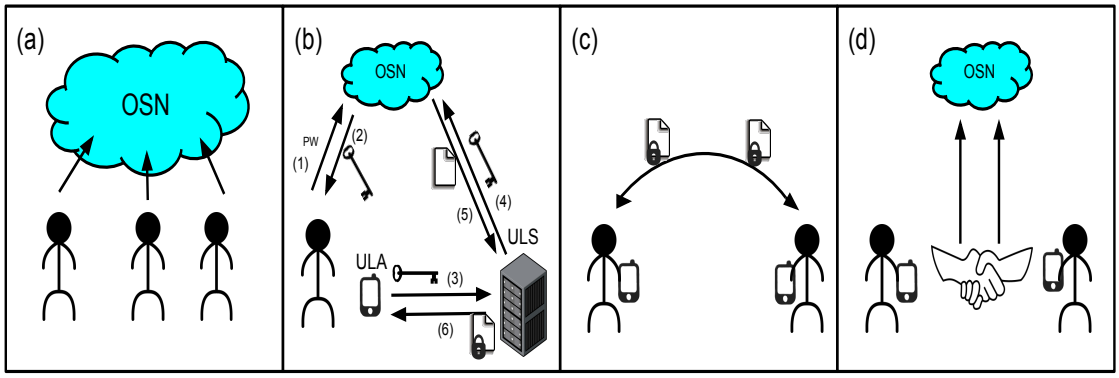


Figure 3.1: Interaction in *UnLinked*: (a) Regular OSN interaction, (b) Setup, (c) Offline, (d) OSN Spillover

3.2.2 Types of Communication in UnLinked

UnLinked involves several types of communication, shown in Fig. 3.1. To start, a user who chooses to use *UnLinked* needs to download and install ULA. Then, a user needs to interact with the OSN and facilitate secure export of her profile to ULS (Fig. 3.1b). This involves user-OSN and ULS-OSN communication. Finally, the main purpose of *UnLinked* is direct communication between off-line users, i.e., a pair of end-devices running ULA (Fig. 3.1c). If two users decide to later connect or become OSN friends, *UnLinked* helps facilitate this interaction, per Fig. 3.1d.

3.2.3 Communication Channels

UnLinked does not restrict the means of communication between the OSN and its users, or between ULS and OSN. We assume that both transpire over the Internet. Off-line interaction between users is assumed to involve a wireless broadcast communication medium that facilitates seamless discovery of peers, e.g., via periodic beaconing. We believe that this is a natural requirement for several reasons. First, per D0, two users need to be physically near each other. Second, virtually all modern personal devices (from laptops to smartphones) communicate over broadcast wireless channels.

Obvious candidates for off-line communication between OSN users are: WiFi, Bluetooth and NFC. Of these, NFC is less appealing than others since it requires the two devices to be very *near* each other. This might be "too close for comfort" in scenarios where users are separated by some distance – e.g., on planes, trains and ships – or whenever they prefer to maintain some physical space and personal privacy. Further, broadcast based user discovery would be impossible via NFC. Both Bluetooth and WiFi are available on a wide range of devices types.

3.2.4 Cryptographic (Privacy) Requirements

Based on design goal D2, we need to support private computation of common factors in OSN profiles of two users. These common factors could include: friends, friends-of-friends, educational and employment histories, as well as various group memberships. Furthermore, D5 requires authenticity of computed common factors.

While online, the OSN allows users to restrict profile information visible to others. Most OSNs only allow friends or friends of friends to view one's profile content. For example, *LinkedIn* requires that two users have some shared profile information before allowing one of them to request a connection. We aim for the same amount of peer privacy but in off-line operation where the OSN itself can not provide it. Thus, we clearly can not stipulate that users simply download and exchange ULS-certified copies of their OSN profiles. Instead, we need to run a set of cryptographic protocols that maintain a comparable (to on-line) level of privacy and allow users to learn only their common factors. This motivates the use of 2-party protocols that perform private set operations, such as Private Set Intersection (PSI) and Private Set Intersection Cardinality (PSI-CA). Furthermore, based on D5, these protocols must operate on user-bound and authenticated input (i.e., profile information), thus guaranteeing authenticity of the result.

3.3 LinkedIn

Most popular OSNs offer users a personal *profile page* to describe themselves: provide information on their place of residence, educational and professional backgrounds, memberships, interests, as well as share photos, videos and other information. Generally, two users can connect to each other by becoming *contacts* or *friends*. Depending on the individual's privacy settings, one typically sees more information about friends, contacts or connections. (We use these three terms interchangeably.) Most OSNs allow users to hide profile information from those who are not direct contacts.

Why *LinkedIn*? Although, based on D7, the general design of *UnLinked* is OSN-agnostic, we needed to make an initial choice of a specific OSN platform. *LinkedIn*'s strong emphasis on validity and integrity of user's connections is important for *UnLinked*. Also, *LinkedIn* appeals mainly to adult professionals who generally tend to value privacy, serious communication and maintaining real world links more than the younger and/or more socially-minded population of Tumblr or Facebook. Furthermore, considering D7 and Section 3.2.1, *LinkedIn* facilitates secure export of user profiles via OAuth and a REST API.

LinkedIn, with more than 250 million users worldwide, is a global OSN that provides a networking platform for professionals. As such, it is widely considered to be more serious or "grown-up" than its more social counterparts, such as *Facebook*. To this end, a typical *LinkedIn* profile page resembles a CV or a resumè, rather than a dynamic scrapbook.

Connections play a major role on LinkedIn. One user gets connected to another by sending an invitation. Such a "direct connection" between two users is established only if the invitee accepts the invitation. The invitee can refuse it and report the inviter as unknown, or the invitation – as spam. An inviter who accumulates many invitation refusals might get her account restricted [Lin13]. This approach constitutes an entry barrier for fraudsters who attempt to join groups of professionals and provides some level of trusted relationships

among users. Moreover, users can benefit from their contact network by getting introduced to new contacts, by their existing contacts. This way, a user has "access" to *second-degree* and even *third-degree* connections.

The main idea is that this "get introduced" approach might lead a user not just to new contacts but possibly to new companies or jobs, as recommended by their connections. Thus, a major goal for *LinkedIn* users is to discover and establish new connections based on criteria such as: common connections, interests, membership and educational or professinal experience.

3.4 Cryptographic Tools

As discussed in Sect. 3.2.4, in order to support off-line interaction, we need privacy-preserving protocols that operate over generic and authenticated input. Furthermore, following design goal D5, a participant's input must accurately reflect its real OSN profile. To accomplish this, we rely on a trusted third party that verifies participant's input and issues a signed certificate of authenticity. ULS functions as this trusted third party: it validates user profile information by direct communication with OSN. We stress that ULS is only needed at setup time and it is not involved in any off-line interaction.

A *setup* phase is performed while A is on-line and connected to the OSN. In this phase, A inputs its profile information a and obtains the corresponding certificate $auth_A$ issued by ULS. This certificate is later used in off-line interactions with other OSN users, i.e., any B that receives $auth_A$ in the off-line phase, can validate a without learning its value.

Generally speaking, our goal in the off-line phase is for A and B to privately compute a function $P(a, b)$ via a two-party protocol P^* that realizes $P()$. To do so, we need to construct $auth_A$ such that a is bound to a specific evaluation of P^* on partial input a. That

is, $auth_A$ bootstraps evaluation of $P(a, \cdot)$. Later, when B presents $auth_B$ to A, continued execution of P^* can only be used to compute $P(a, b)$. The basis for our constructions is that ULS signs the initial message of P^* which is later used as certified input to the off-line phase of the protocol. For this to work, P^* must have the following properties:

1. **Limited-Round:** ULS only signs the initial message sent by each party. All subsequent messages are open to manipulation. By minimizing the number of rounds, we also minimize potential impact of malicious participants who might deviate from the protocol.

2. **Mirrored:** All protocol actions are identical for A and B. This includes cryptographic operations and message transmission order. Thus, allows the same certificate can be used to both initiate and respond to P^*.

3. **Two-Way:** Both parties learn the same correct result $P(a, b)$. We note that this does not hold for two separate instantiations of a one-way protocol, since a malicious participant could provide different input in each one-way instance.

For the two certified private set operation protocols (ATW-PSI and ATW-PSI-CA) used in *UnLinked* we denote the first signed message as $BAS(\cdot)$, and $BAS^{CA}(\cdot)$, respectively. Description of our cryptographic building blocks and notation follows.

3.4.1 Building Blocks

Prime Order Groups

Many of our schemes require a prime order subgroup with a generator g, along with a full domain cryptographic hash-function $H(\cdot)$ with a range of this subgroup. When needed, we use a Diffie-Hellman subgroup defined by primes p and q where $p = tq + 1$, for some integer

t [LK08]. As an instantiation of $H(\cdot)$ we use the SHA-1 hash function, whereby, on input x, $H(x) = (\text{SHA-1}(x))^t \ mod \ p$.

Public Key Signature Scheme

We assume the existence of a public key signature scheme secure against existential forgery under an adaptive chosen message attack [GMR88]. Given a public/private key-pair (PK, SK) we denote such a scheme as a pair of protocols: sign – $SIG_{SK}(\cdot)$, and verify – $VER_{PK}(\cdot, \cdot)$. These protocols operate on message m: $\sigma = SIG_{SK}(m)$ and $VER_{PK}(m, \sigma)$. Verify returns true if and only if σ was computed using knowledge of SK.

Signatures Of Knowledge

We use two popular signature-of-knowledge (SoK) schemes. They operate similar to zero knowledge proofs in that they allow a party to demonstrate knowledge of secret without revealing any additional information (given some shared public knowledge). However, SoK can be performed without interaction, via Schnorr signatures [Sch91]. Specifically, we rely on signatures demonstrate knowledge of (1) a discrete logarithm and, (2) two discrete logarithms (with different bases) being equal. These signatures operate over the same cyclic subgroup as discussed above.

To show knowledge of discrete log of g^x, a party computes SK-LOG as:

$$SKLOG(g^x, g, m) = (c, s) = (h(g^x||g||g^t||m), t - cx)$$

where t is chosen randomly. Upon receipt of $(c, s, y = g^x)$, one can verify that $c = h(y||g||g^s y^c||m)$.

Similarly we can construct an SoK proving equality of two discrete logs with different bases SK-EQ-LOG, i.e., given (g^x, f^y), show that $x = y$. This is done by constructing:

$$SK\text{-}EQ\text{-}LOG(y = g^x, z = f^x, g, f, m) =$$
$$(c, s) = (h(g^x||f^x||g||f||g^t||f^t||m), t - cx)$$

Verification is perfomed as: $c = h(y||z||g||f||g^s y^c||f^s z^c||m)$.

For more details we refer to [ACJT00].

3.4.2 Adversarial Model

Our initial goal is security in the so-called *Honest-but-Curious (HbC)* (aka semi-honest) model. In it, the *adversary* is a protocol participant who follows the protocol while passively trying to learn as much information as possible. However, as discussed in Sect. 3.4.4.4, we can achieve security in the stronger *Malicious* model with some simple extensions. In this model, the *adversary* can arbitrarily deviate from the protocol.

ULS is assumed to be trustworthy, i.e., it honestly and correctly certifies a user's input. We do not consider external adversaries, since standard network security techniques (e.g., TLS and IPSec) can mitigate outsider attacks. Finally, due to the use of hash functions, our constructs are secure in the Random Oracle Model [BR93].

3.4.3 Security Properties

We informally summarize desired security properties:

Correctness. We say that a protocol is *correct* if, whenever both A and B are honest, each outputs $P(a, b)$ at the end of protocol execution, except with negligible probability.

Peer-Privacy. We require secrecy of elements of a and b, unless they appear in the output of $P(a, b)$. For the case of P^* computing a private set intersection, secrecy holds only for elements not in $\{a \cap\ b\}$. In other words, no information beyond $P(a, b)$ is learned by either party.

Pseudonymity. By executing P^*, A and B must not learn each other's identities.

Authenticity. P^* aborts if a or b are not certified by ULS or if either $auth_A$ or $auth_B$ is expired. Thus, the adversary can not learn $P(a, b)$ for any a or b for which it does not hold $auth_A$ or $auth_B$.

Binding. To prevent transferability, an authorization provided to U on input $u = u_1, \ldots, u_n$, must be bound to all of u. That is, U can not transfer or omit any u_i-s.

Output Integrity. If a and b are certified by ULS, then P^* outputs $P(a, b)$. In the HbC model, output integrity is directly implied by authenticity and correctness.

Early Termination Resistance (ETR). If either party aborts the protocol, both must learn nearly[4] equal amounts of information. For an HbC adversary, termination may occur accidentally, e.g., A and B lose connectivity in the middle of the protocol or one of their devices dies. In the malicious model, no protocol can be guaranteed to be *Two-Way*. One party always learns the result first, at which point it can simply abort execution. *ETR* attempts to remedy this imbalance by limiting information conveyed by the final message.

[4]The meaning of *nearly* depends on the specific P^*. The less information revealed in each message, the closer the outputs will be.

3.4.4 Two-Way Private Set Intersection

Authorized Two-Way Private Set Intersection (ATW-PSI) is a protocol between users A and B on respective inputs: ($a = \{a_1, \ldots, a_m\}$, $auth_A$, R_A) and ($b = \{b_1, \ldots, b_n\}$, $auth_B$, R_B):

$$\mathcal{F}_{ATW-PSI}(a,b) \rightarrow \begin{cases} \perp & \text{iff } \neg VER_{PK}(BAS(a), auth_A) \\ & \text{or } \neg VER_{PK}(BAS(b), auth_B) \\ \{x_i | x_i \in a \cap b\} & \text{otherwise} \end{cases}$$

We claim that the combination of the *setup* and *offline execution* phases described below yields a concrete instantiation of the *ATW-PSI* protocol. This construction is based loosely on the One-Way PSI variant from [JL10].

3.4.4.1 Setup

Certification of user input is shown in Fig. 3.2. For user U, let $u = \{u_1, \ldots, u_n\}$ denote the set of U's inputs. ULS, on input of its private key SK, generates a random value $R_U \leftarrow {\mathbb{Z}_p}^*$ (step 1) and signs all hashes of elements in u, masked with R_U (step 2). Both $auth_U$ and R_u are transferred to U (step 3). Note that all exponentiations are *mod p*.

3.4.4.2 Offline Execution

User A on input ($a = \{a_1, \ldots, a_m\}$, $auth_A$, R_A) and user B on input ($b = \{b_1, \ldots, b_n\}$, $auth_B$, R_B) execute the protocol shown in Fig. 3.3. First, A computes a set $Z = \{z_1, \ldots, z_m\}$ where each $z_i = H(a_i)^{R_A}$. A forwards Z along with $auth_A$, to B. This allows B to verify whether A sent valid z_is by computing $VER_{PK}(z_1, \ldots, z_m, auth_A)$ in step (2). In concurrent steps (3) and (4), B computes Y (mirroring A's computation of Z) and forwards Y and $auth_B$ to A. Next, A exponentiates B's y_js with R_A and returns the results to B (step 5). B

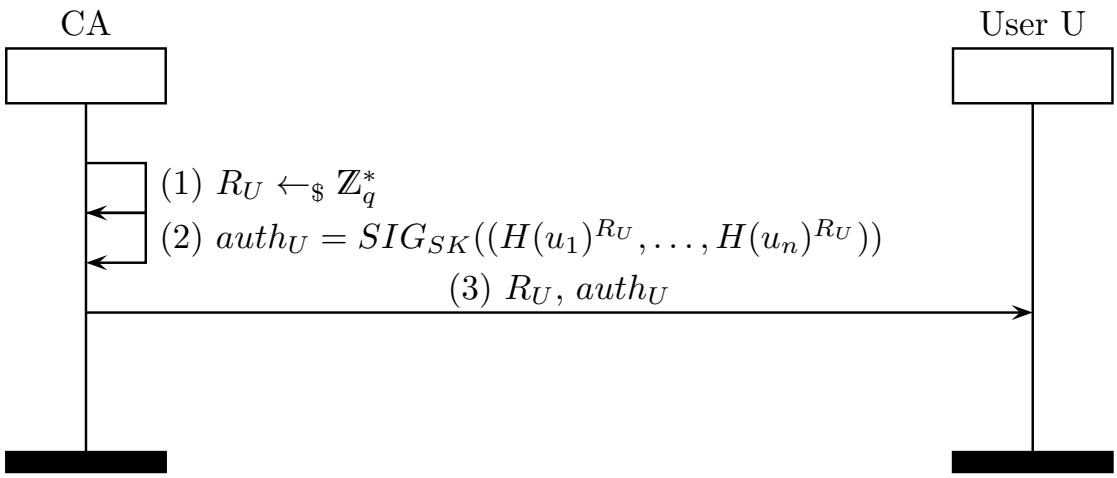


Figure 3.2: Setup

does the same with A's z_is and its own R_B (step 6). Note that protocol steps are executed concurrently when applicable. e.g., (1) and (3), (2) and (4). Also, in steps (5) and (6), each element sent by A is followed by one element sent by B. In step 7, both A and B output the intersection of a and b, i.e., a set of elements a_t (resp. b_s) where $y_s^{R_A} = z_t^{R_B}$ for $1 \leq s \leq m$ and $1 \leq t \leq n$.

3.4.4.3 Security Considerations

We claim that ATW-PSI is a peer-private, pseudonymous instantiation of PSI, with binding in the presence of malicious adversaries in the Random Oracle Model [BR93]. Furthermore, ATW-PSI provides protocol execution integrity in the HbC model. We sketch the proof below. Since ATW-PSI is *Mirrored*, we assume w.l.o.g. that B plays the role of the adversary B^*.

Correctness follows trivially from the protocol description. If both parties are honest, both compute:

$\{H(a_1)^{R_A R_B}, \ldots, H(a_m)^{R_A R_B}\} \cap$

$\{H(b_1)^{R_A R_B}, \ldots, H(b_n)^{R_A R_B}\}$

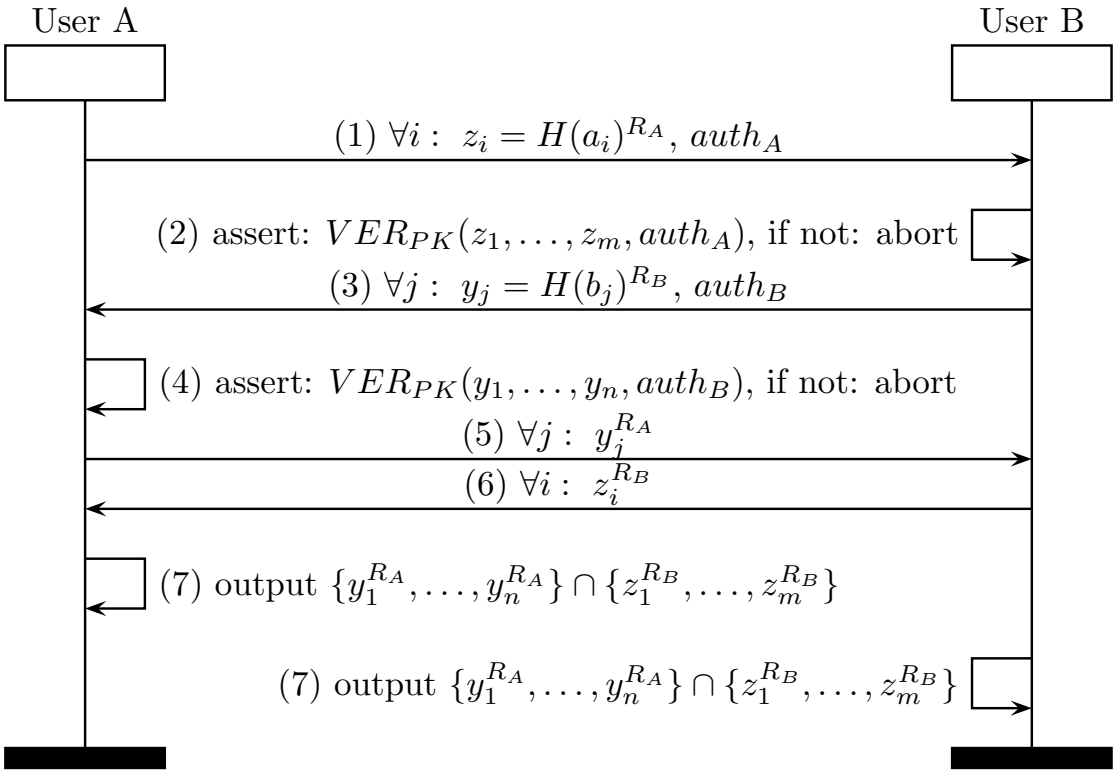


Figure 3.3: Offline Execution Phase

This is equivalent to: $\{H(a_i)^{R_A R_B} | s.t. \ \forall i \ \exists j \ a_i == b_j\}$. Thus, both parties output the intersection $a \cap b$.

Peer-Privacy. For an HbC adversary, privacy is provided by the One-More Gap Diffie Hellman Assumption (OMG-DH) [Fre05]. Informally, this provides indistinguishability of exchanged elements of the form $H(x)^R$, even with adversarial access to a Diffie-Hellman oracle. Only matching elements are learned in step (7). All privacy arguments (in the HbC model) from [JL10] apply to ATW-PSI with minor adjustments.

However, our protocols offer privacy in the *malicious* model with no further modifications. Since it can not deviate from the protocol without forging ULS's signature, B^* can only manipulate its response to A's initial message, in step 6. However, this message is completely

de-coupled from any of A's responses. Thus, no matter what it sends, B^* learns no any additional information.

Pseudonymity. *Peer-Privacy* implies privacy of A's and B's identities during one protocol instance. However, further information can be learned from multiple protocol executions. As long as A re-uses the same $auth_A$, all off-line interactions are initiated with this value, and B can easily correlate multiple encounters with A.

However, whenever it is on-line, A can always contact ULS and request a fresh certificate, i.e., a new pseudonym $auth'_A$, thus preventing B from linking future interactions with $auth'_A$ to any prior interactions involving $auth_A$. This is because any two protocol transcripts, both involving A on input a, with different blinding factors R_A and R'_A, are indistinguishable.

As discussed in Sect. 3.6.3, pseudonymity can be strengthened if a user obtains a batch of pseudonyms for each off-line epoch. In a single on-line interaction with ULS, A can request a batch of certificates and use each one as few or as many times as it wants. Clearly, if each $auth_A$ is only used once, pseudonymity transitions into anonymity.

Authenticity follows directly from unforgeability of the underlying signature scheme SIG. If either $auth_A$ or $auth_B$ does not verify in steps (2) or (4), execution is aborted.

Binding. Each $auth_U$ is bound to the entire set u. Therefore, binding is trivially achieved due to unforgeability of SIG.

Output Integrity. In HbC, this follows directly from security of SIG used by ULS and correctness of ATW-PSI. However, in the malicious model, A can convince an honest B that the protocol output is any subset of $a \cup b$.

3.4.4.4 Malicious Security

In its current form, ATW-PSI does not provide *Integrity of Output* in the malicious model. To achieve this stronger security, we need to modify steps (5) and (6) in Fig. 3.3 to force users to adhere to the protocol. This can be done using techniques similar to [JL10], i.e., by introducing two Schnorr-based signature of knowledge (SoK) constructs: SK-LOG and SK-EQ-LOG [ACJT00], described in Section 3.4.1.

First, we modify *setup* to include g^{R_a} within $auth_A$. During *Offline Execution*, A provides a single (Schnorr) signature SK-LOG proving knowledge of R_a. Later, in step (5), A provides a proof of correctness that shows, via SK-EQ-LOG, that each exponentiation ${y_j}^{R_a}$ is performed with the same R_a supplied earlier in g^{R_a}. These measures demonstrate that: (1) A indeed uses the same R_a in all exponentiations of $y_j s$, and (2) A uses the R_a as signed by ULS and included in $auth_A$. For B's part, Step (6) is modified similarly.

Early Termination Resistance (ETR). Even with the Integrity feature, a malicious participant can abort the protocol at any time. We counter this by enforcing concurrent execution of steps (5) and (6). Specifically, messages between A and B are "interleaved", i.e., each party alternates between transmitting and receiving a specific set element ${z_i}^{R_b}$ or ${y_j}^{R_A}$. The two will only swap once the computation has been verified, using SoK. This way, at any point during protocol execution, parties compute a equal portion of the intersection. A malicious participant learns at most one additional element; thus, its advantage is very low.

Clearly, ETR does not come for free. The modified protocol incurs $O(max(n, m))$ rounds, instead of the previous two. Although in some scenarios this added complexity might make the protocol impractical, recall that *UnLinked* users are expected to be near each other and transmission latency is therefore expected to be negligible.

3.4.5 Two-Way PSI Cardinality

As the privacy and usability of *UnLinked* can be greatly enhanced by additional crypto primatives, we present here an extension of PSI-CA form [DGT12] to construct a linear-time Two-Way Private-Set Intersection Cardinality protocol with Authenticity (ATW-PSI-CA). We follow a similar procedure to ATW-PSI with minor modifications to *setup* and *offline* phases. Resulting protocols are shown in Fig. 3.4 and Fig. 3.5 respectively. They rely on two random permutations Π, and Π'.

ATW-PSI-CA provides the same security as ATW-PSI, however, at additional cost. Unlike ATW-PSI, authorizations ($auth_U$) cannot be reused. Instead, U must obtain from CA a distinct $auth_U$ for each future off-line interaction. In Sect. 3.7 below, we argue that, in the context of *UnLinked*, this is a small price to pay for better privacy.

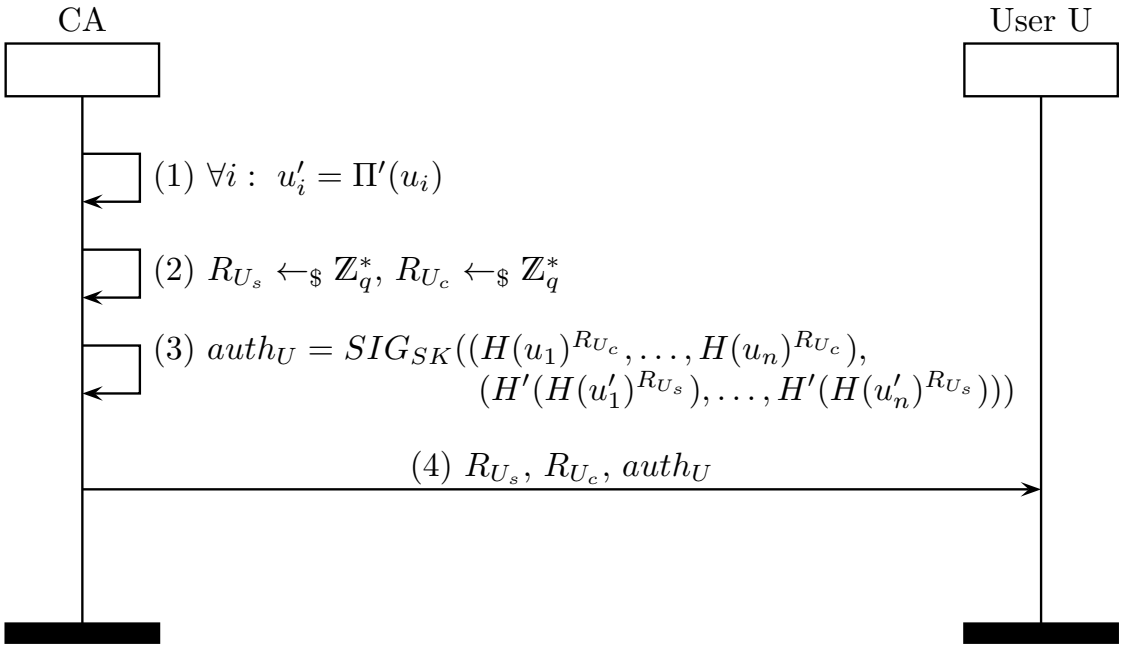


Figure 3.4: Setup w/ Cardinality

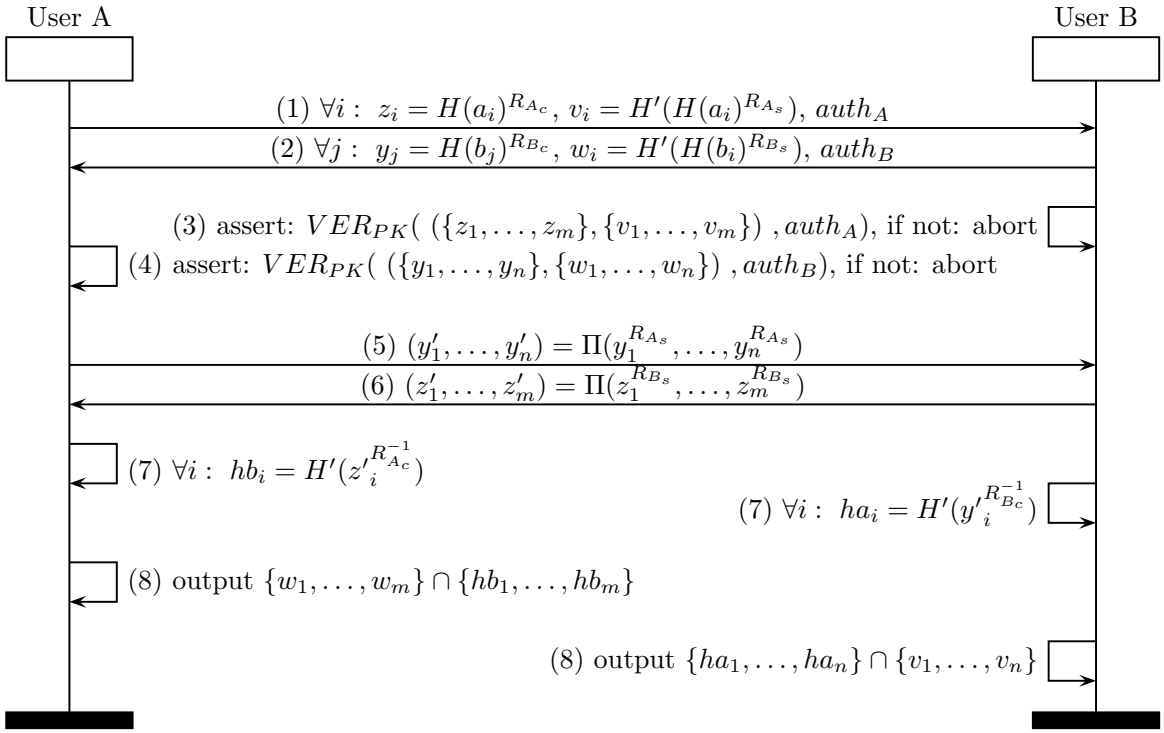


Figure 3.5: Offline Execution Phase w/ Cardinality

3.5 System Architecture

UnLinked includes two phases: Setup and Off-line. This section describes basic system assumptions and the details of each phase.

3.5.1 Requirements

Although *UnLinked* is OSN-agnostic, there are a few requirements:

- In order to support meaningful off-line interactions, OSN profile information must contain at least a unique owner's user id and a list of connection's ids. Also, OSN must provide the ability to securely export a user profile once authorized by the user. This is offered – via OAuth2 API [Har12] – by several popular OSNs, e.g., *LinkedIn*,

Facebook, Twitter, and Google+. In our setting, an OAuth2 interface allows ULA and ULS to communicate to OSN on behalf of the user.

- ULS requires a public key signature scheme $[SIG_{SK}(\cdot), VER_{PK}(\cdot, auth)]$ with a public/private key-pair: $[PK, SK]$. ULS retains no state information about its interactions with any ULA or OSN.

- ULA needs access to a broadcast medium, in order to support nearby peer discovery. ULA is also responsible for storing all user's private cryptographic information, including BAS_U. In addition, ULA must trust ULS's PK; this is provided by including PK in the ULA installation package.

3.5.2 Setup Phase Details

The goal of this phase is for ULA to obtain enough information to be able to later operate independently from OSN and ULS. Since a typical OSN user's profile can change fairly often, e.g., on a daily basis, we need to ensure that users have up-to-date profile information. For this reason, ULS issues BAS_U with a relatively short lifespan, e.g., one week in the current implementation, as described in Sect. 3.6.4. Setup proceeds as follows:

1. Via local web browser, Alice logs in to OSN, retrieves and stores an OAuth token *tok*. Alice's password **is not revealed** to ULA.

2. Alice's ULA presents *tok* to OSN, retrieves her profile P and stores it locally.

3. Alice's ULA presents *tok* to ULS.

4. ULS relays *tok* to OSN

5. OSN sends Alice's profile P.

6. ULS parses P into basic components:

   $[c = \text{connections}, \text{identity}, \text{misc}]$

   where identity consists of the user-name or identifier as well as a profile picture, if any. The misc field is partitioned into OSN-specific sub-fields, i.e., education, employment and residence.

7. ULS computes and sends to ULA: $\sigma_{id} = SIG_{SK}(\text{identity})$, along with $\sigma_1 = SIG_{SK}(BAS(c))$, $\sigma_2 = SIG_{SK}(BAS(\text{misc}))$ and the corresponding random blinding factors Ru_1, Ru_2[5].

3.5.3 Off-line Phase Details

This phase handles communication between peer ULAs. Once connected, ULAs privately compute common information of their profiles. Depending on user policy, this may entail multiple rounds of communication and executions of both ATW-PSI (defined in Sect. 3.4.4) and ATW-PSI-CA (defined in Section 3.4.5). We now describe the interaction between two nearby ULAs: Alice and Bob, assuming that the former starts the interaction.

1. Without involvement of the actual human users (i.e., in the background) Alice and Bob emit broadcast "pings" or beacons at fixed time intervals[6], advertising their membership in *UnLinked*.

2. Alice detects Bob's broadcast and responds with a PID. PID selection is discussed in Section 3.6.2 below.

3. If Bob wishes to continue communication, it responds with its own PID.

4. Using standard techniques (TLS, in our case), ULAs establish a secure channel.

[5]If requested by ULA, ULS uses $BAS^{CA}(\cdot)$ (Blinded Attribute Set for ATW-PSI-CA) instead of $BAS(\cdot)$
[6]Typical interval between pings is 30-60 secs.

5. ULAs execute a series of ATW-PSI-CA protocols, privately computing the size of their common connections and miscellaneous fields. Comparing connections means exchanging σ_1, $BAS(c)$, followed by a series of short computation and transmission rounds. See Sect. 3.4.4, and Sect. 3.4.5 for more details.

6. Next, ULAs perform one or all of the following, depending on their mutual desire to continue interaction, either by explicit user action, or by policy. Their choices are conveyed across the secure channel.

   (a) ULAs perform a series of ATW-PSI protocols to compute the actual set intersection of their profiles, analogous to computation and communication in Step 5. Each instance of ATW-PSI is performed over a different type of data, e.g., connections, employers and educational institutions.

   (b) ULAs exchange authenticated identities of the form (identity, σ_{id}). Upon receipt, identity is validated and results are displayed to the user.

   (c) ULAs exchange messages via a simple chat interface.

   (d) If both decide to later connect through OSN, each ULA stores the peer's identity. At a later time, when it is on-line, ULAcan use the OSN to request a connection to the peer.

3.5.4 Notification Policy

As mentioned earlier, during the off-line phase, ULAs perform a series of cryptographic tests, each on one of various profile fields. A ULA determines explicitly which tests are conducted and, for every test, based on its results, whether it wants to continue interacting with a particular peer. However, such fine-grained control over every interaction can be cumbersome. We envision numerous scenarios where large numbers of OSN users, all with something in common, are in physical proximity. For instance, on a university campus, most

users will have the same employer (or educational institution) and several connections in common. In such settings, most users would rather not to be notified of every nearby peer. In other settings (e.g., on a plane, or simply far from home) a user may wish to be notified of any nearby peer even with very little in common between their profiles.

To increase flexibility, we introduce the notion of personal *policy*. Each policy includes a set of conditions for ULA to initiate each of the following actions:

1. Execute a more revealing protocol. i.e., ATW-PSI instead of ATW-PSI-CA.
2. Alert the user to the presence of a peer.
3. Exchange identifying information, e.g., name, image, or OSN profile id.
4. Send a message to a peer.
5. Receive and display messages and requests to exchange ids.

Criteria for these actions can be any combination of the following:

1. A threshold or a range based the size of the intersection of connections or any *misc* sub-field.
2. Presence or absence of a specific value in the intersection.

Note that criteria of the first kind can be applied using only the result of ATW-PSI-CA. Whereas, the second type of criteria relies on successful ATW-PSI execution for that field.

While our policy language for *UnLinked* is flexible, ULA is pre-configured with a few default policies. All such policies require explicit user actions to exchange identifying information and send messages. Furthermore, criteria for notifying the user of a message or identity

exchange are identical to that for ULA to be alerted to the presence of a nearby peer. This simplifies effects of policy to the user.

The most permissive supported policy, *Open*, allows ULA to run ATW-PSI with any *Un-Linked* peer. This policy is useful for maximizing off-line interactions. A slightly more restrictive policy, *Low*, is designed for use away from one's typical locations, e.g., on travel. In this case, the user is alerted of a nearby peer's presence if any of the following conditions are satisfied:

1. At least one friend in common, and one school or employer.
2. At least three friends in common.
3. A common current employer or current academic institution.

We also provide a third default policy, *Medium*, designed for use at everyday locations, such as work-place or school. In this case, having an employer or a school in common is not very meaningful. Instead, we add extra weight to the value of past employers and connections. Users are notified if any of the following occurs:

1. At least three connections in common, and at least one common employer or academic institution, other than the current one.
2. At least five connections and one institution in common.
3. More than seven connections in common.

Preventing SPAM: User policy also controls the amount of communication with peers. If Alice and Bob have different policies, Bob might wish to contact Alice but not vice versa. Suppose that Alice is not alerted of Bob's presence. Should Bob's contact attempts be transmitted to Alice? This decision is based upon Alice's policy. If desired, Alice can specify

a different threshold for receiving messages from a peer than that for being initially alerted to the peer's presence. Without this feature, Bob could bombard Alice's ULA with spam messages or connection requests simply by acquiring many fake OSN identities.

3.6 Discussion and Extensions

We now discuss some system considerations and extensions.

3.6.1 Minimizing Irrelevant Connections

In some settings, ULA may encounter the same peer multiple times. Being notified of each such encounter is burdensome to the user and costly in terms of device resource consumption. To this end, ULA has a mechanism for preventing repeated interactions. However, a given user's profile information might change frequently and to ensure new information is considered in future encounters this process is time-dependent. To this end, the prevention mechanism operates on two levels. First, ULA maintains a list of recently encountered device MAC addresses. Entries in this list have a lifespan of one day. We consider this to be an acceptable granularity, since user profiles rarely change dramatically within a single day. Second, each ULA broadcasts its own public identifier PID. Upon connecting to a new device, ULAs exchange $PIDs$. If a PID has been seen recently, communication is terminated. When an off-line interaction completes, the peer device address is added to the *avoid list*.[7]

In addition to avoiding recently seen peers, we consider a likely scenario where two users who are already each other's connections discover each other anew and attempt to connect. Clearly such unnecessary interactions should be avoided. For this purpose, we provide a simple mechanism that allows users to learn, and to be optionally alerted, when current

[7] $PIDs$ are refreshed when new profile information is downloaded.

connections are nearby. As part of every interaction with ULA, ULS inserts a "dummy" tuple corresponding to that ULA into the set of connections returned to ULA. Upon interacting with a peer, this tuple will appear in the result set if and only if the two are already connected.

3.6.2 Authenticated Channels

There are many ways to select $PIDs$. To easily establish an authentic channel we take advantage of the trusted ULS. The basic idea is that each PID is the public portion of a Diffie-Hellman key exchange. Specifically, for a user U, $PID = g_u^R$. To fully prevent man in the middle attacks, ULS includes this PID in all authorizations returned to U. This effectively binds all certificates to the corresponding user.

3.6.3 Unlinkability

For a privacy conscious user, *UnLinked* can also operate in an *unlinkable* mode. In it, ULA contacts ULS and receives a new authorization along with a new private key R_u, whenever possible. This provides the user with a unique pseudonym linked to the specific R_u and allows the user to control the degree of unlinkability. Of course, other factors influence both privacy and performance. To be completely unlinkable, a user might wish to avoid redundant connection optimizations discussed previously. Using the mechanism for detecting pre-existing connections would significantly lower the level of anonymity to their connections, and could trivially reveal their identity in some cases. Furthermore for each pseudonym, ULA needs to reset the list of previously discovered peers. Otherwise unwillingness to participate in an interaction may itself be used to link pseudonyms. Finally, ULA may need to make device configuration changes dependent on the medium, e.g., change MAC addresses.

Nonetheless, even with these safeguards, peers always learn the outcome and input size of cryptographic protocols. This information could be used to help identify the other user. Exact severity of this de-anonymization is dependent on the uniqueness of one's contact graph and common profile traits.

3.6.4 Freshness of Credentials

In order to operate offline, *UnLinked* relies on authenticated profile information. Since profiles can change with arbitrary frequency, ULAs should be assured of freshness of peer profiles. Otherwise, misbehaving users with stale or outdated information could claim connections they no longer have. Intuitive ways of dealing with this problem rely on either time-stamped signatures or revocation lists. Since *UnLinked* operates off-line and aims to minimize overhead, distributing periodic revocation information is not feasible. Instead, all ULS signatures include a global timestamp and a relatively short validity interval of one week. We believe that this is sufficiently long for ULA to regain connectivity to ULS. At the same time, it is short enough to minimize the impact of malicious peers. To support this feature, we assume that each ULA's clock is loosely synchronized with that of ULS. This is a reasonable assumption for most modern devices, even when operating off-line.

3.6.5 Detecting Misbehavior

Since ATW-PSI and ATW-PSI-CA do not offer output integrity from a malicious peer by default, *UnLinked* provides an alternative to handle malicious behavior. This is achieved by allowing ULS to audit communication traces from offline interactions. A ULA records every protocol transcript. Later, when it re-gains connectivity, ULA sends a transcript to ULS along with its current profile. ULS audits every message in the transcript and determines whether peers behaved correctly. ULS can then blacklist misbehaving users or

take other measures to prevent further cheating. However, auditing interaction transcripts imposes considerable storage and computation burden on ULS and exposes ULA's off-line interactions to ULS. Consequently, auditing is an optional feature, disabled by default. Only a user who suspects fraud should submit transcripts for auditing.

3.7 Implementation & Evaluation

We developed and benchmarked a fully functioning prototype of *UnLinked*, consisting of an Android application implementing the functionality of the ULA and a Ruby on Rails application implementing ULS.

For the signature scheme ULS computes "attached" RSA signatures (those that include the signed data in the signature) using PKCS7 implemented over OpenSSL. Currently, ULA supports a wide range of communication technologies: WiFi Direct, Bluetooth, and WLAN based Network Service Discovery. Any subset of these can be used concurrently. The current prototype only supports ATW-PSI without extensions for malicious security. For the purposes of benchmarking, ULA was tested on two Nexus 5 smartphones communicating over Bluetooth. Each phone has a 2.26GHz processor and 2GB of RAM. ULS was tested on a desktop running Ubuntu 12.10, with an Intel i7-3770 3.4GHz quad-core CPU and 16GB of RAM.

Performance: *UnLinked* was tested on several input sizes: 10, 100, 1000, and 10,000 friends. As expected, protocol execution time was linear in the number of friends. For 100 friends, ULS signature computation completed in 0.83s, and used 25.2kb of storage. Also, each ULA completed the Offline Phase of the ATW-PSI protocol in under 1 second, with 526kb exchanged. See Tab. 5.2 for detailed results. We note that, even with 1,000 friends, information returned by ULS in the setup phase only consumed 243kb. One could easily batch

100 of such signatures to interact with unlinkability off line, without significant storage impact. Similarly, authorizations for ATW-PSI-CA can be batched without significant storage costs. Finally, round-trip times between devices was, on average, only 7.86ms. When using ATW-PSI with interleaving this amounts to less than 1 second of overhead for the input size of 100 friends.

Table 3.1: Evaluation of *UnLinked*.

Input Size	Setup Time (s)	Signature Size (kb)	Offline Time (ms)	Offline Band. (kb)
10	0.0739	3.41	143	9.2
100	0.837	25.2	508	75
1000	8.57	243	8790	2941
10000	86.4	2420	14530	7337

3.8 Operational Algorithims

Below are the specific algorithms used in *UnLinked* in greater detail.

Setup: The setup phase assumes that OSN user Alice already installed *client* on her mobile device:

1. Alice logs into OSN.
2. Alice invokes *client* which communicates to *server*.
3. Alice authorizes *server* to export her profile from OSN.
4. *server* authorizes Alice for off-line interaction by signing a blinded representation of Alice's profile attributes (e.g., her connections), referred to as a Blinded Attribute Set (BAS_A).

The setup phase is repeated periodically to refresh authorizations.

Off-line: The off-line phase assumes that Alice and Bob have previously completed their respective setup phases:

1. Alice and Bob discover each other via location-limited beaconing by respective *client*.
2. Using BAS_A and BAS_B, respectively, Alice and Bob execute a cryptographic protocol that yields either the number, or the set of, common factors (e.g., shared connections).
3. Alice and Bob optionally choose to reveal their signed OSN identities to each other.

On-line: The on-line phase assumes that Alice and Bob have already performed an off-line phase and revealed identities to each other:

1. Alice issues an OSN contact request for Bob.
2. At some point, Bob accepts the request.

Chapter 4

Secure Pattern Matching

This chapter details existing secure computation protocols for pattern matching and their uses in today's society. It is based on my publication *Blindfolded Data Search via Secure Pattern Matching* published in IEEE Computer written along with Karim El Defrawy [EDF13]. While the presentation has changed no extentions have been made.

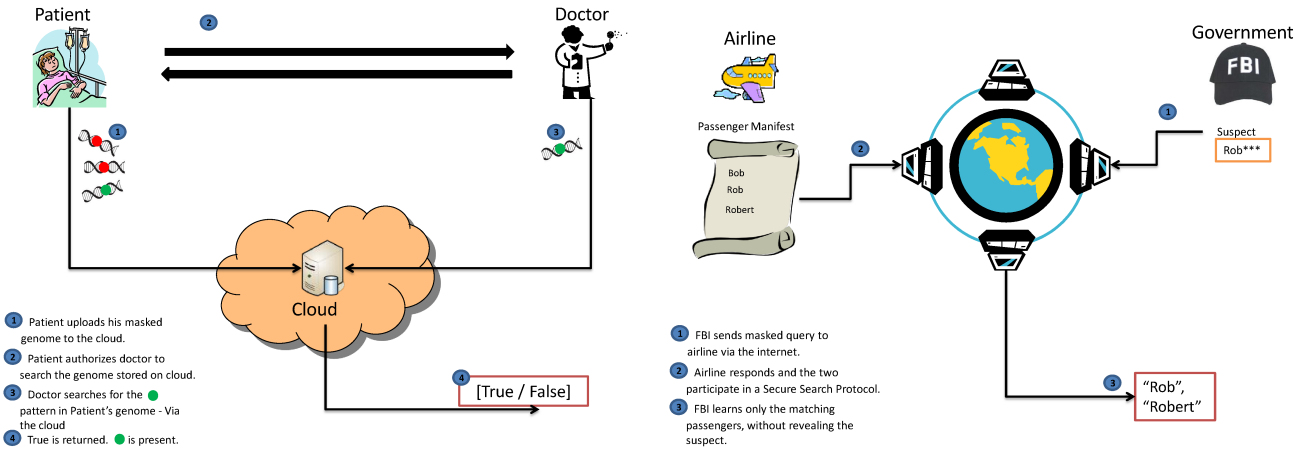
Secure computation protocols have recently attracted considerable attention. This may be attributed to the fact that various organizations and users are adopting software and computation as a service, instead of building their own computation infrastructure. Secure computation protocols in general allow two or more distrusting parties to collectively perform computation that requires inputs from the parties and deliver outputs to some, or all, of them. Typically, two properties have to be guaranteed during operation of such protocols: (1) correctness of performed computation, and (2) privacy of inputs and outputs of parties. This chapter describes some of the best known techniques to securely search data while guaranteeing the two general properties required in secure computation. This is achieved by securely and efficiently performing one of the most fundamental types of computation often encountered in computer science: pattern matching.

Roadmap: The need for blindfolded searching of data is highlighted by several applications; we describe some representative application domains in Section 4.1. We proceed to provide background on the general problem of pattern matching then that of secure pattern matching in Section 4.2. We define the pattern matching problem with its different variations that are often encountered in modern applications; we then describe relevant security and adversary models. Section 4.3 follows with an overview of some recent protocols that efficiently perform secure pattern matching. We conclude with a discussion of open problems and future research directions in Section 4.4.

4.1 The Need for Blindfolded Searching of Data

We begin by describing two application domains that highlight the need for secure and privacy-preserving searching of data and how secure pattern matching can help address them. In the considered notional applications an information requester will be able to privately query an information provider's database and retrieve only the records that match its query pattern, without revealing the query or its results to the provider. The provider would like to prevent revealing any information about its database other than matched records.

1. **Sharing of Data in Intelligence and Law Enforcement Communities:** Over the past decade, issues have arisen regarding the desire to share data with local law enforcement, state and local governments, first responders, and international governments versus potential privacy concerns. A recent example of such discord was the European Court of Justice ruling in May 2006 that ordered the cessation of passenger name record data-sharing with the U.S. Though the data required was relatively innocuous, it took two years to implement a new agreement and there are still privacy advocates within Europe who oppose the sharing of data without stronger data privacy safeguards [KE09]. More difficult is sharing sensitive intelligence or law enforcement



(a) Medical Diagnosis/Treatment (b) Intel Sharing

Figure 4.1: A patient visits a doctor who privately examines his genome for treatment (a). FBI privately examines a passenger list for a suspect with name like "Rob" (b).

data with the tens of thousands of state, local, and tribal governments within the USA. Balancing security concerns with information sharing remains a top priority for several intelligence and law enforcement agencies. Developing solutions that simultaneously provide strong protection for privacy and security while enabling access to the latest intelligence and law enforcement databases remains an open problem. Secure pattern matching as will be described in detail later, can enable secure evaluation of an expressive set of queries to search such data including: exact matching, substring and approximate matching, and range queries for numerical data. See Figure 4.1(a) for a demonstrative example.

2. **Sharing of Medical and Health Data**: The standards for "Privacy of Individually Identifiable Health Information" establish a set of national standards for the protection of individuals' health information. The U.S. Department of Health and Human Services (HHS) issued such standards to implement the requirement of the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [CHLR96]. These standards address the use and disclosure of individuals' health information by organizations subject to the standards, as well as standards for individuals' privacy rights to understand and

control how their health information is used. A major goal of the standards is to ensure that individuals' health information is properly protected while allowing the flow of health information needed to provide and promote high quality health care. The standards aim to strike a balance that permits important uses of information, while protecting the privacy of people who seek care and healing. Secure pattern matching can ensure privacy of such personal health information while at the same time allowing doctors, hospitals and other health organizations to use such information in medical care. A notable example highlighted in the security community [KM10b] is to securely and privately search personal DNA sequences. It is obviously important to prevent leakage of an individual's DNA sequencing data because it may, among other things, indicate pre-exposure to certain health risks which may cause insurance providers to deny them coverage (or even increase their premiums).

4.2 Background and Overview of Secure Pattern Matching

We first describe the problem of insecure pattern matching and its variations. We then describe various adversary and security models to be considered when developing a secure pattern matching protocol.

4.2.1 Insecure Pattern Matching

Pattern matching is fundamental to the field of computer science. It finds applications in searching databases, networking and security and recently in the context of bio-informatics and DNA analysis [KM10b]. Pattern matching has been extensively researched, resulting in several efficient algorithms and approaches to solve various insecure flavors thereof, e.g.

[Tsa06, KR87]. Nonetheless, there is currently no single algorithm that performs well in all cases and is efficient in both its speed and memory usage in addition to handling exact, approximate matches and wildcards. There are many interpretations of the pattern matching problem, the most common being the following: given an alphabet Σ, a text $T \in \Sigma^n$, the *exact pattern matching decision problem* requires one to decide if a pattern appears at all in the string. In the case of the *exact pattern matching search problem* the requirement is a little different, it is finding all indicies i of T (if any) where P occurs as a substring. If $\bar{T}_i$ denotes the m-character substring of T starting at position i, the output should be the set $\{i|\bar{T}_i = P\}$. Throughout this chapter we use DNA as our alphabet, i.e., $\Sigma = \{A, C, G, T\}$, to simplify illustrations.

In addition to the exact matching problem, the following generalizations are often encountered:

1. *Pattern matching with single character wildcards:* there is a special character, $* \notin \Sigma$, that is not part of the alphabet and could match any single character of the alphabet. This special character can be repeated several times in the pattern P indicating that the positions where it is present could contain any of the alphabet's characters. The output in this case should also be the set of indicies: $\{i|T_i = P\}$. Using such a "wildcard" character allows one pattern to be specified that could match several sequences of characters. For example the pattern "$TA*$" , would match any of the following words in a text: TAA, TAC, TAG, and TAT

2. *Substring pattern matching*[1]*:* In this version a parameter $k \leq m$ is fixed. A match for P is found whenever there exists in T an m-length string that differs in $m-k$ characters from P (i.e., has Hamming distance $m - k$ from P). For example, the pattern TAC has $m = 3$. If $k = 2$, then any of the following words would match ($*$ indicates that any character from the alphabet could be substituted): $*AC$, $T*C$, or $TA*$. Note that

[1] Also sometimes called approximate or threshold pattern matching.

results from single character wildcard matching will also be captured with substring matching. Referring to the previous example, a substring matching with the pattern TAC and $k = 2$ will also capture all occurrences of $TA*$.

4.2.2 Security and Adversary Models

A protocol that performs secure pattern matching on a text T held by a server and using a pattern P held by a client can be defined as an interactive two-party protocol. Typically, both parties are assumed to be probabilistic polynomial time algorithms. The view of a party in this interactive protocol is defined as its private input and its private randomness as well as the protocol transcript.

The informal security requirements [2] of secure pattern matching dictate that the party holding the text learns nothing, except the upper-bound on the length of the pattern, while the one holding the pattern only learns either a binary (yes/no) answer for the decision problem or the matching positions (if any), and nothing else.

The security of secure computation protocols, and thus secure pattern matching, is typically demonstrated using the so-called ideal/real world simulation paradigm. In order to prove security using that paradigm one imagines what properties a protocol should have in an ideal world. In the ideal world all parties send their input to a trusted third party (TTP) [3] that computes the functionality in question and returns the output to each party. The TTP is assumed to be incorruptible and thus by definition the ideal world is secure. An adversary can not influence the inputs of any parties other than the ones it corrupts, neither can it learn outputs of any other parties. A protocol is considered secure if the real (the constructed) protocol has similar views as in the corresponding ideal world.

[2] For a formal definition of secure pattern matching check [BEM+12].
[3] Also called ideal functionality.

The main adversary models considered when constructing secure protocols per paradigm above are:

1. **Honest-but-Curious Adversaries:** Honest-but-curious (HBC) adversaries do not deviate from the steps prescribed by the protocol specifications. HBC adversaries may try to violate privacy of the other party by storing exchanged messages, snooping on messages and trying to analyze whatever data they receive to infer more information than they are supposed to. Such adversaries are often also called passive or semi-honest adversaries, because they passively listen to messages and try to violate privacy and confidentiality but do not actively modify, fabricate or delete messages in order to launch any kinds of active attacks.

2. **Malicious Adversaries:** Malicious adversaries are expected to launch different types of passive and active attacks and are not assumed to fear being caught cheating. They are expected to deviate arbitrarily from the prescribed protocol and are not trusted to perform correctly whatever computation is required from them. They may send garbage and/or inconsistent data, and even replay messages. Malicious adversaries are typically assumed to not be stealthy, i.e., they are not trying to hide the fact that they cheat. Protecting against malicious adversaries provides a very strong level of security, but usually can only be achieved using expensive cryptographic primitives such as zero-knowledge proofs and commitment schemes.

3. **Other Adversary Models:** Other models capture adversaries that are more powerful than HBC but less powerful than malicious ones. Notably are covert adversaries, which may deviate arbitrarily from the protocol specication in an attempt to cheat, but do not wish to be caught doing so [AL07].

Protocol	Client BW	Server BW	Total Client CC	Total Server CC	WC SM	Security Models
1 (Sec.4.3.1)	$O(mnk^2)$	$O(mnk^2)$	$O(mnk \log^2 k)$	$O(mnk \log k)$	Yes	M/HBC
2 (Sec.4.3.2)	$O((n+m)k^2)$	$O((n+m)k^2)$	$O(mnk \log^2 k)$	$O(mnk \log k)$	Yes	M/HBC
3 (Sec.4.3.3)	$O((n+m)k^2)$	$O((n+m)k^2)$	$O(mnk \log^2 k)$	$O(mnk \log k)$	Yes	M/HBC
4 (Sec.4.3.4)	$O(mk')$	$O(nk+mk')$	$O(n+mk \log^2 k)$	$O((n+m)k \log^2 k)$	No	1S/HBC

Table 4.1: Comparison of pattern matching protocols.

Comparison assumes protocols requiring public-key operations use ElGamal encryption with a k-bit security parameter. BW = Bandwidth, CC = Computation complexity, M = Malicious model, HBC = Honest-but-curious model, 1S = One-sided Simulation. WCSM = wild card and substring matching. All protocols run in O(1) rounds.

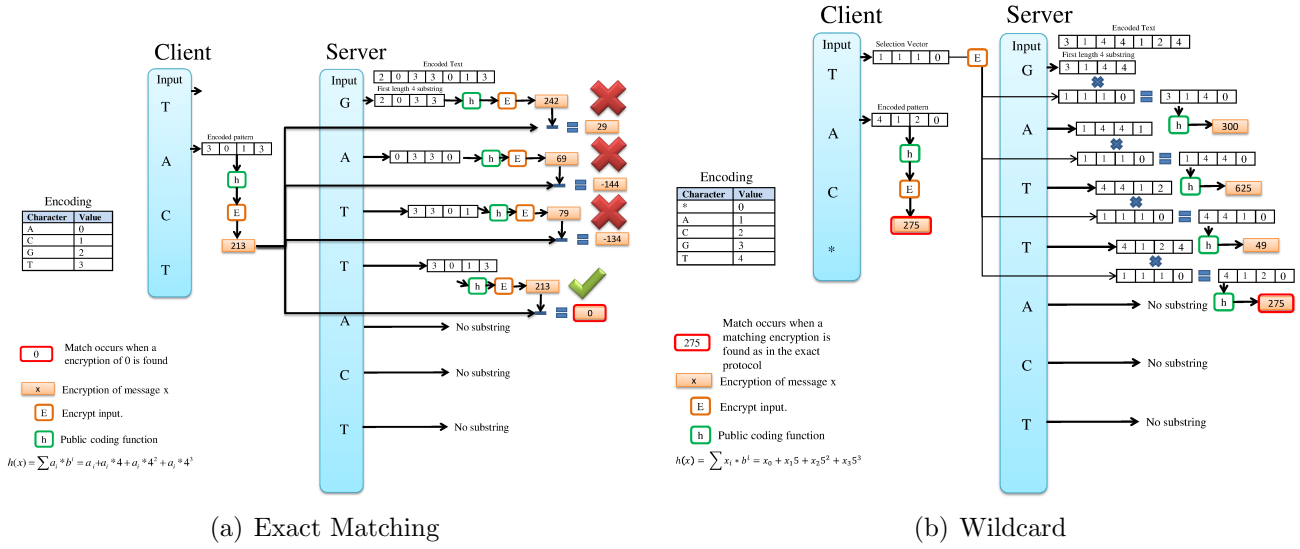
4.3 Overview of Secure Pattern Matching Protocols

We overview below the design and performance of some recently developed secure pattern matching protocols. Recall that the secure exact pattern matching setting consists of a server holding text $T \in \Sigma^n$ and a client holding $P \in \Sigma^m$. We let $\bar{T}_i$ represent the m length substring rooted at position i of the text. Further, the wild card pattern matching problem allows an additional wild card character denoted as $*$.

4.3.1 Protocol 1: Integer Comparison Based Pattern Matching

The authors in [HT10] present protocols, secure against malicious adversaries, that can compute variants of the pattern matching problem. The security of their protocols are based on the security of (additive) ElGamal encryption, a well known vetted cryptosystem. The protocols come in three flavors to solve a specific variant of the pattern matching problem. The first two variants rely on turning the pattern, and each of the m length sub-strings of text into an encryption of a single integer via a function $h(x)$. These encryptions are then compared obliviously. Each of these two variants can support alphabets of any size, and without loss of generality we assume that the alphabet is a subset of the integers. Thus, each sub-string is encoded as a base $|\Sigma|$ integer. i.e. $h(x) = \sum_{i=0}^{m} x_i * |\Sigma|^i$.

For exact matching, the client encodes the pattern as $P' = E(\sum_{i=1}^{m} P_i * |\Sigma|^{i-1})$. The server computes for each $\bar{T}_i$, $\bar{T}_i{}' = E(\sum_{j=0}^{m-1} T_{i+j} * |\Sigma|^j)$. P' is then compared against each $\bar{T}_i{}'$ to identify a match. This can be performed using the homomorphic properties of (additive) ElGamal encryption. $\forall i$ the two parties compute $(P')^{-1} * \bar{T}_i{}'$, which is an encryption of 0 if and only if the two are equal. The two parties then jointly check, in zero-knowledge, if this is indeed an encryption of 0, and thus a matching position in T. See Figure 4.2(a) for a detailed example. In order to ensure correctness under a malicious adversary, both parties perform all computation. Further, correctness of the inputs and the computation is verified using zero-knowledge proofs. The exact matching protocol requires $O(n+m)$ cryptographic computation and communication.



(a) Exact Matching (b) Wildcard

Figure 4.2: Client queries using the exact matching from [HT10] "TACT" (a) and wild card matching"TAC*" (b). Server's text remains "GATTACT" with a single match in both cases.

The authors of [HT10] also present a slightly modified protocol to handle wildcard matching. The main idea is that wildcards in the pattern will be selectively removed before being encoded as in the previous protocol. The client replaces wildcard characters ($*$) in the pattern with 0. The client then sends an m length vector of encrypted bits, called the Selection Vector, that can be used to effectively "select" the non wildcard characters from

each $\bar{T_i}$. Thus the client sends

$$sv_i = \begin{cases} E(0) & \text{if } P_i = * \\ E(1) & \text{otherwise} \end{cases}$$

This vector is verified by the server and the encrypted text T is returned. The client then computes, verifies, and randomizes for the ith position of the jth $\bar{T_j}$

$$\hat{T_{i,j}} = T_{j+i-1}^{sv_i}$$

Again, due to the homomorphism, any character paired against a wildcard in the pattern is an encryption of 0, as is the wildcard in the pattern. With wildcards removed the protocol continues as before, computing $P' = E(\sum_{i=1}^{m} P_i * |\Sigma|^{i-1})$, and $\bar{T_i} = E(\sum_{j=0}^{m-1} \hat{T}_{i+j} * |\Sigma|^j)$. See Figure 4.2(b) to see an example run on small DNA string.

For every possible matching position in the text the protocol performs $O(m)$ cryptographic-operations to remove the wild cards. This results in an overall complexity of $O(nm)$ communication and computation.

The final protocol securely computes the secure approximate matching problem, for bit strings only, in O(nm) computation and communication. This limitation can severely affect its practical applications[4]. The protocol relies on securely computing the hamming distance of P and each $\bar{T_i}$. A match is found if this distance is above a specified threshold τ. The crux of the protocol is the computation of the hamming distance. As before, the server sends the encrypted text as $T_i = E(t_i)$. The client computes and sends for the ith position of the jth sub-string, $\Pi_{i,j} = T_{j+i-1}^{P_i}$. Effectively, this is an encryption of $P_i \wedge T_{j+i-1}$. They then both compute $X_{i,j} = T_{(j+1-1)} * P_i * 2\Pi_{i,j}$. This is the encryption of $P_i \oplus T_{j+i-1}$, or the hamming distance in binary.

[4]A pattern in an encoded binary string does not necessarily correspond to a pattern in the original text

4.3.2 Protocol 2: Fast Fourier Transform (FFT) Based

The authors in [Ver11] solve the pattern matching problem by utilizing the Fast Fourier Transform (FFT). The key observation is that one can use an FFT algorithm to quickly compute sums of products of entries in a vector. [Ver11] derives a protocol for exact, approximate, and wildcard matching. The wildcard matching protocol runs with $O(nlog(m))$ exponentiations, $O(nm)$ multiplications, and $O(n)$ communication. It works by first mapping wildcards in the text and pattern to 0 and every other character to a positive integer, as in Protocol 1. Then, the protocol solves for the sum of squared error between the pattern and each possible matching sub-string of the text. Both securely compute $\forall i \in [n-m+1]$

$$\sum_{j=1}^{m} P_j * T_{i+j-1}(P_j - T_{i+j-1})^2 = \sum_{j=1}^{m} P_j^3 * T_{i+j} - 2P_j^2 * T_{i+j-1}^2 + P_j * T_{i+j-1}^3$$

Notice that if P_j or T_{j+i-1} is a wildcard the jth term in the sum will always be 0. Similarly, if $P_j = T_{j+i-1}$ the jth term is also 0. Thus the entire sum evaluates to 0 if and only if the pattern matches the ith sub-string, i.e. the error between strings is 0.

All of these sums can be computed securely in $O(n \log m)$ using a protocol that securely computes polynomial multiplication via convolution (FFT). To see how this works, first we look at the first term in the sum $P_j^3 * t_{i+j-1}$ We treat each of the P_i^3 as coefficients of an m degree polynomial $A(x)$. Similarly, we let t_i be the coefficients of an n degree polynomial $B(x)$. If we can quickly determine the coefficients of $A(x)B(x)$ we will be able to calculate $P_i^3 * T_j \, \forall i, j$. Using these values we easily compute the appropriate terms in the sum. We can compute the other two terms similarly. For an in-depth explanation of polynomial multiplication using FFT see [Ver11] which provides such a protocol secure in the presence of malicious adversaries. This protocol also relies on the security of (additive) ElGamal encryption.

4.3.3 Protocol 3: Matrix Multiplication Based Pattern Matching

The authors in [BEM+12] construct a two party protocol, 5PM, that can securely calculate the exact, wildcard, and approximate pattern matching problems for any alphabet. The protocol is secure against a malicious adversary provided the underlying homomorphic encryption scheme is semantically secure. The crux of the protocol is a reduction from pattern matching to a non-FFT based matrix multiplication. Their algorithm requires $O(nm)$ cryptographic operations, and $O(n+m)$ communication. It is worth noting that this is the only protocol mentioned with an accompanying implementation. The protocol for wildcards is explained in detail below.

5PM first preprocesses the pattern to obtain a *Character Delay Vector* (CDV) for each letter in the alphabet. These CDVs are used to compute, for each character in the text, the position where a matching pattern could possibly end. A counter at this location is incremented. If a match occurs in the text, than all m characters in the matching $\bar{T}_i$ would have caused the counter to increment. Thus, if at the end of the protocol any counter equals m a match exists.

Insecurely the algorithm is as follows. For each character in the alphabet a m length CDV is created initialized to all 0's. Next, for each character P_i in the pattern a 1 is set in the corresponding CDV indicating a possible ending $m-i$ characters away. That is, $\forall i$ $CDV(P_i)[m-i] = 1$. Observe, for all CDVs and all i, the ith position is 1 in exactly one CDV. Next, the Activation Vector (AV), a length n array of counters, is created and initialized to 0 element wise. The AV is then computed as

$$\forall i \in [1..n], j \in [1..m] \quad AV[i+j] = CDV(T_i)[j] + AV[i+j]$$

Finally, we output i if $AV[i] = m$. See Figure 4.3 for CDVs of the pattern $TACT$.

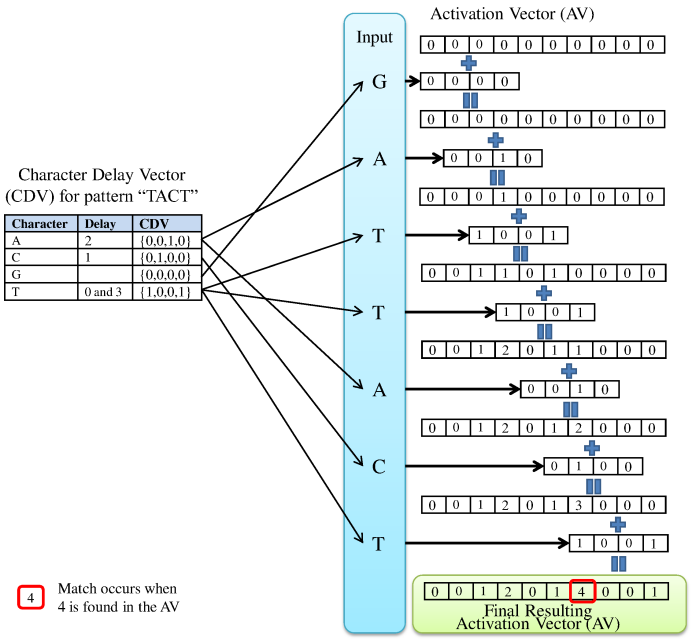


Figure 4.3: Character Delay Vectors for "TACT" and their application to the Activation Vector during processing.

The algorithm can be easily extended to handle wildcard pattern matching. Upon CDV creation wildcard characters are not processed. Further, in the final matching phase we check $AV[i]$ against $m - k$ where k is the number of wildcards in the pattern.

This same algorithm can be computed using a combination of three linear operators, equivalent to matrix multiplication. Briefly, ColSum(M) computes the sum of each column and stores the result in a length n vector. Cut(M) removes both the left m columns and the right m columns. Stretch(M) shifts the ith row of M right i columns, and places 0 in empty locations. The resulting matrix has an extra n columns. See Table 4.4 for an example. For a more detailed explanation of these operations consult [BEM+12].

To use these operators the text is first transformed into a binary $n \times |\Sigma|$ matrix M_t such that $M_t(i, T_i) = 1$. Essentially, each row in M_t encodes a 1 in the location corresponding to the correct letter in Σ. We also encode the CDVs as a matrix, by concatenating each CDV together. We then calculate the matrix $M_{t(CDV)}$ corresponding to a copy of the appropriate CDV in row i for each text position T_i. This is a simple multiplication, $M_{t(CDV)} = M_t * M_{CDV}$. We then Stretch($M_{t(CDV)}$) to center the ith CDV at position (i, i).

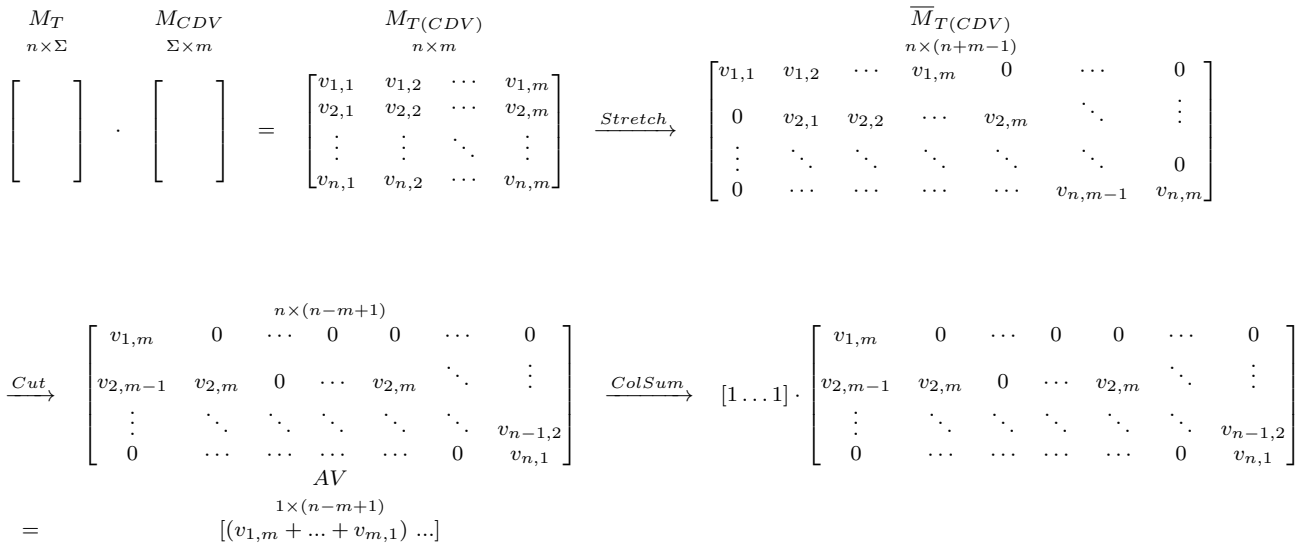


Figure 4.4: Insecure Pattern Matching Expressed as Linear Matrix and Vector Operations.

Finally we compute the activation vector by cutting the extraneous rows where a match can not occur, and summing the columns to calculate the counter at each position i. i.e. $AV = ColSum(Cut(Stretch(M_{t(CDV)})))$. As before, i is returned if $AV[i] = m$.

Using this construction, creating a secure protocol is straight forward. The server computes and sends an encryption of his text matrix M_t. The client computes and sends an encryption of the CDV matrix M_{CDV}. Using an additive homomorphic encryption scheme both the server and the client compute an encryption of AV as $E(AV) = ColSum(Cut(Stretch(M_{t(CDV)})))$, which due to linearity of matrix multiplication is computable if either M_t or M_{CDV} is encrypted. The two parties then subtract m from each element in $E(AV)$ and jointly test for an encryption of 0, indicated a match. [BEM+12] include variants of this idea secure against HBC and malicious adversaries.

4.3.4 Protocol 4: Garbled Circuit Based Text Processing

For some applications, additional text processing may be required once a match is found. In this setting the server and client will perform a secure protocol in which the client learns a function $f(P, T_i, i)$ for only indicies i where the pattern matches, $P = \bar{T}_i$. The authors in [KM10b] develop a protocol to securely compute this functionality by relying on two cryptographic primitives keyword search and garbled circuits [Yao82]. Briefly, keyword search allows a client to query for matching keywords from a server's database, returning an arbitrary payload upon match. Garbled circuits allow for arbitrary secure function evaluation via boolean circuits, however it requires computation linear in the circuit size. They use keyword search to solve the exact pattern matching problem, and garbled circuits to compute the generic functionality f on the matched results. First, they solve the exact pattern matching problem, by requiring the pattern P to be a query, and each $\bar{T}_i$ to be a keyword By customizing the payload of the servers keywords they can transmit information allowing the client to securely compute $f(\bar{T}_i, i, P)$. A variety of information could be sent for specific f's, however the authors observe that, in most cases, f is small enough to be easily computed using Yao's garbled circuits [Yao82]. First, the server transmits u garbled circuits to the client. u being an upperbound on the number of matches of P in T. Next, the server sets the payload for each keyword $\bar{T}_i$ to the keys corresponding to his input of f, $(\bar{T}_i; i)$. In this way the client can add his input, P, and securely compute f for each match.

The novelty in this protocol is it's ability to quickly compute a large class of text processing functions while still remaining efficient. The protocol runs in $O(n + m)$ exponentiations for the keyword search phase, and with $O(u)$ gates. For circuits secure against the HBC adversary each gate requires several symmetric key operations. While powerful, we note that it has limitations. Primarily, it provides weaker than full malicious security. It also cannot handle substring or wildcard pattern matching (Although inefficient extensions exist).

In addition, for efficient preprocessing of most circuits the server must know the value of m before meeting with a client.

4.4 Open Problems

Despite the recent focus on developing secure pattern matching protocols, there still remain several open problems. First, all of the secure pattern matching protocols, except for 5PM, have not been implemented and thus little is known about their concrete timing and memory requirements; such analysis is critical to assess the applicability of such protocols to real-life scenarios. Second, most secure pattern matching protocols are either designed for the HBC adversary model or the malicious adversary model. To the best of our knowledge, there are no secure pattern matching protocols for other adversary models, such as covert adversaries. As explained earlier, the covert adversary model is much stronger than the HBC model and provides adequate security for real world application, typically without requiring expensive cryptographic primitives such as zero-knowledge proofs. Third, as multimedia and images constitute a significant fraction of today's (and future) data, it is natural to ask whether one can devise more efficient secure pattern matching protocols for multidimensional data instead of adopting protocols that are designed for one-dimensional data. As an example of multidimensional data, biological data, e.g., proteins, may require matching in multiple dimensions. Finally, with the recent emphasis on streaming and "big" data it is natural to require secure generic pattern matching to operate efficiently on large amounts of streaming data with constant memory and reasonable processing and timing constraints. This area still remains largely unexplored.

Chapter 5

Size- and Position-Hiding Private Substring Matching

This chapter addresses the problem of size and position hiding private substring matching and its use in privacy-preserving genomics. It is based on my publication *Secure Genomic Testing with Size-and Position-hiding Private Substring Matching* published in the Workshop on Privacy in the Electronic Society written along with Emiliano De Cristofaro and Gene Tsudik [DCFT13]. While the presentation has changed no extentions have been made.

Whole Genome Sequencing (WGS) is a revolutionary technology that determines the complete genetic blueprint of any organism. In the context of human genomics, WGS is expected to foster significant progress in the quality of healthcare [CM01]. In particular, genetic features and mutations are being increasingly and more effectively studied in relation to treatment of – as well as predisposition to – diseases and genetic conditions, such as Alzheimer's, diabetes and various types of cancer. Availability of fully sequenced genomes makes such testing seamless and low-cost, since complex operations can now be executed in software, in a matter of seconds [GW09]. This fuels the race toward cheaper, faster, and more accurate

sequencing technologies. Naturally, progress in genomics and WGS prompts numerous immediate and anticipated benefits. At the same time, it amplifies security, privacy and ethical concerns that stem from unprecedented sensitivity of genomic data. The human genome contains detailed and extremely personal information, e.g., susceptibility to somatic and mental conditions as well as ethnicity and ancestry. Therefore, potential disclosure triggers fears of genetic discrimination, aka "eugenism". Moreover, due to its hereditary (cumulative) nature, the human genome encodes information beyond the individual: it also contains information about one's siblings, parents as well as other ancestors and descendants.

A genome uniquely identifies its owner, which makes anonymization and de-identification techniques ineffective [GMG+13, Mal05, H+08, W+09, ZPL+11]. Security and privacy issues in genomics have been studied by security experts [BBD+11, Mal05] as well as biologists, ethicists [SOJH09, H+08, GSMG11] and policy-makers [Pre12]. We refer to [ADHT13] for a comprehensive overview.

Risks of disclosure motivate the need for secure storage and testing of human genomes. In particular, there is a need for genetic testing by physicians and/or laboratories without full access to individuals' genomes. In an idealized future setting, genetic tests should only disclose the required minimum amount of information. At the same time, genomics and biotechnology companies often treat test details as trade secrets, since they represent valuable intellectual property [Gen12b, Gen12a]. For instance, Myriad Genetics recently attempted to patent two human genes, which, if mutated, are associated with significantly increased risk for breast and ovarian cancers [Pol13]. The legality of these patents was recently challenged and the US Supreme Court has declared them invalid [Wol13]. This decision was based on the premise that, owing to its natural occurrence, no part of a genome is patentable. We believe that, because of this landmark decision, the commercial sector will have no choice but to keep its test specifics (e.g., genetic markers) proprietary. Thus, while the outcome

of a genetic test might be made available to the patient and/or the testing laboratory, test specifics would be kept confidential by the latter.

These challenges extend to the emerging field of *personalized medicine*, i.e., the practice of tailoring diagnoses, pre-symptomatic examinations, and treatments to the precise genetic makeup of individual patients. Already today, a number of companies (e.g., 23andme, i-gene and Knome) provide customers with detailed reports about their predisposition to diseases and conditions. Several drugs (e.g., for treating cancer, HIV and leukemia) are coming to market accompanied by related genetic tests, that are necessary to assess correct dosage and/or expected effectiveness [Bur12, Abb03, PB08]. Again, while details of such tests might be proprietary, patients are strongly dis-incentivized from submitting their entire genomes for testing.

Motivated by the above discussion, this chapter explores efficient cryptographic protocols for genomic testing that satisfy aforementioned privacy requirements. Specifically, we focus on a setting where one party, *server*, holds a copy of his digitized genome and the other, *client*, holds a (possibly non-contiguous) substring that might occur in a certain location of *server*'s genome. The specific problem at hand is: *how to allow server to test for the presence of client's substring at a certain location in his genome, such that server learns nothing about the substring (including its position and its size), while client learns nothing about the genome?*

We address this problem by building a novel cryptographic primitive, called **Size- and Position-Hiding Private Substring Matching (SPH-PSM)** Despite its genomics-based motivation, SPH-PSM is appealing in any substring-testing setting where size and position (and, clearly, the contents) of the partial substring should be concealed, since both values can leak information about the *nature* of the test.

We also describe a prototype of SPH-PSM and demonstrate its practicality via a thorough performance evaluation, which shows that many genomics tests can be conducted today in under a minute.

Chapter Organization: The next section presents the problem statement. Section 5.1.2 describes and analyzes the SPH-PSM protocol. Next, Section 5.2 presents efficient instantiations of SPH-PSM and introduces several optimizations. Finally, Section 5.3 discusses further extensions and variations.

5.1 Problem Statement

The quest for Predictive, Preventive, Participatory, and Personalized (P4) medicine [HG09] has been one of the main motivating factors in genomics research. With the advent of low-cost sequencing technologies, the natural next step is to provide physicians and testing facilities with computational means to query, correlate, and analyze entire digitized genomes [You12].

One prominent challenge is where and how to store a digitized genome, i.e., 3.2 billion letters. This issue is rife with privacy and trust considerations. As noted in [BBD+11], individuals should ideally retain ownership of their sequenced genome and selectively allow partially trusted third parties (such as physicians, clinicians and testing facilities) to query it.

Privacy and ethical concerns prompt the need for secure genomic tests that offer simultaneous confidentiality of the individual's genome and test specifics. Since data obfuscation and anonymization tools are ineffective in the genomic context [GMG+13, Mal05, H+08, W+09], secure computation techniques are needed to realize privacy-preserving versions of these tests, whereby only the test result is disclosed to one or both participants. This presents some challenges: First, secure computation techniques must contend with the sheer size of the genome, which makes it crucial to maximize pre-computation and minimize protocol

input size. Furthermore, each current genetic test needs to be mapped to a function with output conveying nothing beyond the test outcome. For example, a testing facility may want to check for the presence of a few DNA markers in a patient's genome to determine correct dosage for a blood thinning drug: a privacy-preserving version of this test would disclose nothing beyond whether this patient has these exact markers.

Specifically, the test should not leak:

1. ***Position(s)*** of tested markers,

2. ***How many*** markers are being tested, and

3. The ***subset*** of matched markers, in case of an overall negative result.

In the genomic setting, where the number of current genomic tests is relatively small, the size- and position-hiding are particularly important. This is because disclosure of the number or positions of markers could allow the adversary to infer test specifics. Satisfying aforementioned three requirements is an open problem, which we address in this chapter.

5.1.1 Definitions & Notation

We now introduce a technique for private genomic testing, whereby a testing facility checks for the presence of a substring or a pattern in a patient's genome, such that the latter learns the outcome and no other knowledge is obtained by either party. We do so by introducing a cryptographic primitive called Size- and Position-Hiding private Substring Matching (SPH-PSM). After defining privacy requirements, we present a generic construction based on additively homomorphic encryption and analyze its security.

Our **notation** is reflected in Table 5.1.

$a \leftarrow_{\$} \mathcal{A}$	Variable a chosen uniformly at random from $\mathcal{A}$
Σ=$\{A, G, C, T, -\}$	DNA Alphabet ('$-$' denotes deletion)
$p = \{\Sigma\}^m$	Potential substring held by *client*, of length m
$t = \{\Sigma\}^n$	String held by *server*, of length n
$i \in [1, m]$, $j \in [1, n]$	Indices of characters in p and t, respectively
p_i, t_j	Elements in positions i and j, of p and t, respectively
$t[j : x]$	Substring in t starting at t_j, of length x
s	Presumed starting location of p in t
λ	Security parameter
AddHomEnc	An IND-CPA additively homomorphic cryptosystem
PK, SK	*server*'s private and secret key for AddHomEnc
$E(\cdot), D(\cdot)$	Encryption (with PK) and decryption (with SK) in AddHomEnc
$\mathbb{G}$	Plaintext group of E
$q = ord(\mathbb{G})$	The order of the plaintext group
$E(a) \cdot E(b) = E(a + b)$	Multiplication of two ciphertexts resulting in addition of two plaintexts
$E(a)^c = E(a \cdot c)$	Exponentiation resulting in multiplication of plaintext by constant
$\perp$	No output
$\equiv^c$	Computational indistinguishability

Table 5.1: Notation used throughout the chapter.

Definition 5.1 (Size- and Position-Hiding Private Substring Matching (SPH-PSM).)**.** *A protocol involving two parties: Client, on input* $((p = p_1, ..., p_m), s)$*, and Server, on input* $(t = t_1, ..., t_n)$*:*

$$F_{SPH\text{-}PSM}((p, s), t) \rightarrow (\perp, b), \textit{where } b = \begin{cases} 1 & \textit{iff } t[s : m] = p \\ 0 & \textit{otherwise} \end{cases}$$

Correctness. If both parties are honest and run on correct input (as above), at the end of the protocol, *server* outputs 1 iff $t[s : m] = p$ and 0 otherwise (except with negligible probability).

Notation used throughout the rest of the chapter is reflected in Table 5.1.

Adversarial Model. We use standard security models for secure two-party computation and assume the Honest-but-Curious (HbC) model. The term *adversary* refers to insiders, i.e., protocol participants. Outside adversaries are not considered, since their actions can be

mitigated via standard network security techniques. Protocols secure in the HbC adversarial model assume that all parties faithfully follow all protocol specifications. However, during or after protocol execution, any party might (passively) attempt to infer additional information about the other party's input.

We argue that, in the genomic testing setting, HbC security is sufficient. Genomic testing usually takes place in a medical lab or a physician's office and we therefore expect some degree of trust between the patient and the testing lab. Thus, we envision no incentive for participants to arbitrarily deviate from the protocol. Also, due to their sensitivity, genomic tests can be audited and there might be legal consequences for malicious behavior by either party. Furthermore, in the event of a testing facility breach and/or data loss, privacy of the genome holder remains guaranteed.

The HbC model is formalized by considering an ideal implementation where a trusted third party (TTP) receives the inputs of both parties and outputs the result of the desired function. Security in this model requires that, in the real implementation of the protocol (without a TTP), each party does not learn more information than in the ideal implementation. We introduce simulation-based privacy definitions below.

NOTE: Recall that, in our setting, *Client* plays the role of the testing laboratory and *Server* is the individual in possession of a digitized genome.

Client'**s Privacy.** Let $View_B((p, s), t)$ be a random variable representing *server*'s view during the real execution of SPH-PSM. We say that SPH-PSM guarantees *Client*'s privacy if there exists a Probabilistic Polynomial Time (PPT) algorithm B^* such that, given *server*'s output b, the following holds:

$$\{B^*(t, b)\}_{((p,s),t)} \equiv^c \{View_B((p, s), t)\}_{((p,s),t)}$$

That is, for all possible inputs, with the knowledge of the output bit b and *server*'s input, B^* can efficiently simulate the view of *server*.

Server'**s Privacy.** Similarly, let $View_A((p,s),t)$ be a random variable representing *client*'s view during the real execution of SPH-PSM. We say that SPH-PSM guarantees *server*'s privacy if there exists a PPT algorithm A^* such that, given $n = |t|$:

$$\{A^*((p,s),n)\}_{((p,s),t)} \equiv^c \{View_A((p,s),t)\}_{((p,s),t)}$$

Although we require knowledge of text length for efficient simulation, in many applications this variable is public or fixed, e.g., 3.2 billion for human genomes.

Additively Homomorphic Encryption. We assume the existence of an efficient additively homomorphic public-key cryptosystem with indistinguishability under CPA attacks (IND-CPA). To ease presentation, we denote it as AddHomEnc. Recall that there are several AddHomEnc instantiations, including: Paillier [Pai99], Dramgard-Jurik [DJ01], as well as the Additively Homomorphic ElGamal variant [ElG85].

For each instantiation application of homomorphic operations vary slightly. As described in Table 5.1, we denote the homomorphic addition operation as multiplication of ciphertexts. Further, we denote multiplication by a constant by the exponentiation of a ciphertext.

5.1.2 Proposed Construction

We now present an SPH-PSM protocol that involves *server* on input a string t (i.e., a genome), and *client* who holds a substring p (of length m) that might occur at location s in t. We later extend this protocol to support multiple starting locations. At the end of the

interaction, *server* learns nothing about *client*'s substring, *not even its length or intended position*, while *client* learns the test outcome, i.e., $b = 1$ iff $t[s : m] = p$ and $b = 0$ otherwise.

Protocol. Figure 5.1 illustrates the protocol. It relies on an additively homomorphic cryptosystem AddHomEnc, composed by $KeyGen(1^\lambda)$, $E(\cdot)$, and $D(\cdot)$. It assumes that *server* has previously generated a keypair (PK, SK) for AddHomEnc, as part of *KeyGen*.

First, *server* encrypts, under its own public key, each nucleotide of the genome and sends *client* the resulting ciphertexts. Starting at position s, *client* homomorphically adds each encrypted nucleotide to the encryption (under PK) of the inverse of each substring character. If the corresponding plaintexts match, the product of the two ciphertexts is an encryption of zero, due to the homomorphic property of AddHomEnc. To guarantee privacy, each product is re-randomized by exponentiating it to a random value. Then, *client* multiplies all products so that, if the substring is found in *server*'s genome, the result would still be an encryption of zero. Otherwise, it corresponds to an encryption of a random number. Finally, this cumulative ciphertext (denoted as *acc*) is sent to *server*. Upon decryption, *server* learns the test outcome: positive if the ciphertext decrypts to zero and negative otherwise.

Complexity. *Communication* overhead amounts to $O(n)$ ciphertexts, i.e., $\approx$ 3.2 billion encrypted nucleotides. Although this might seem overwhelming, we discuss how to reduce online costs in Section 5.3.4. On the other hand, *computational* complexity is minimal, since the only operation that involves all n nucleotides is encryption of *server*'s genome. This needs to be done only once, ahead of time, for all possible tests. In fact, it could even be done by the sequencing lab, at sequencing time. Whereas, *online* computational complexity is linear in the size of the substring, i.e., $m = |p|$. Specifically, *client* needs to perform $O(m)$ operations (where $m << n$), while *server*'s overhead is constant — one decryption.

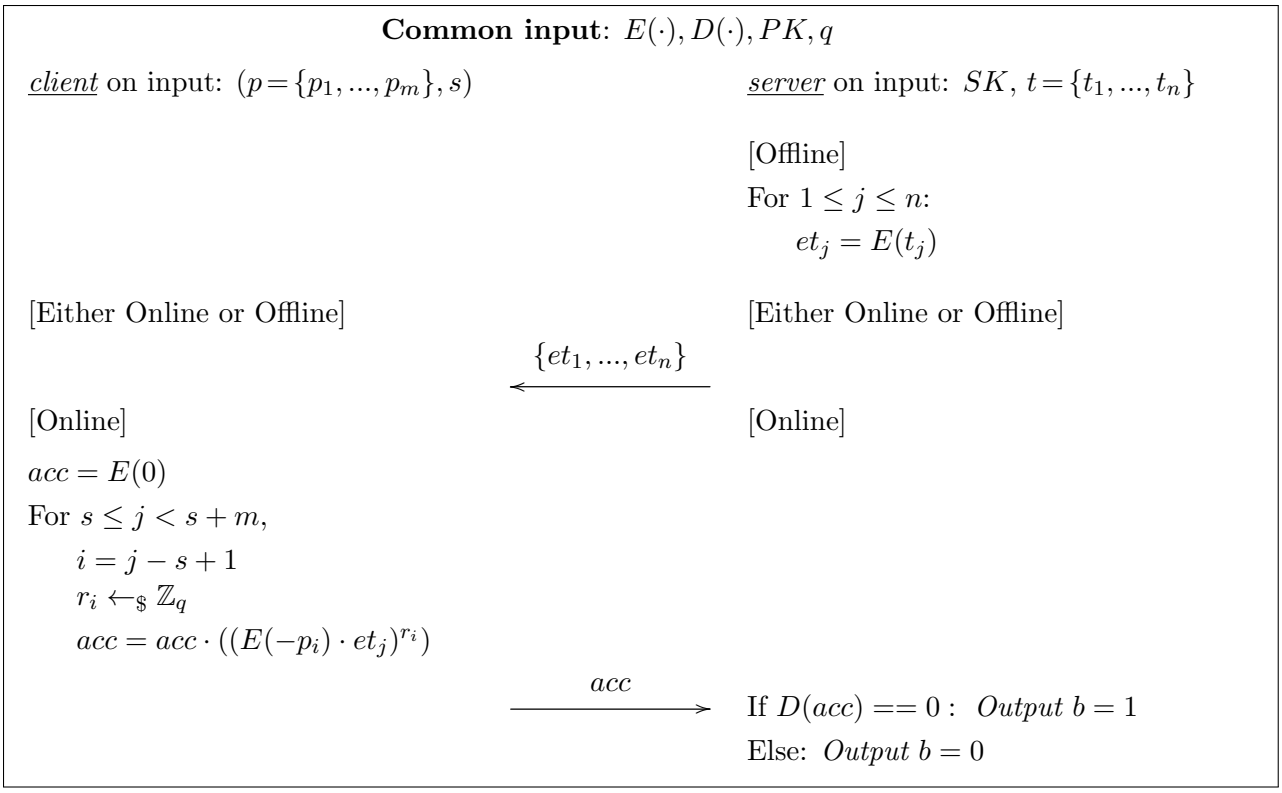


Figure 5.1: Base-line SPH-PSM Protocol.

Finally, *client* only performs computation on ciphertexts et_j for $s \leq \ j < s + m$. All other ciphertexts can be discarded. Also, *client*'s computation can start as soon as et_s is received. Therefore, *client* requires only a negligible amount of local *storage*.

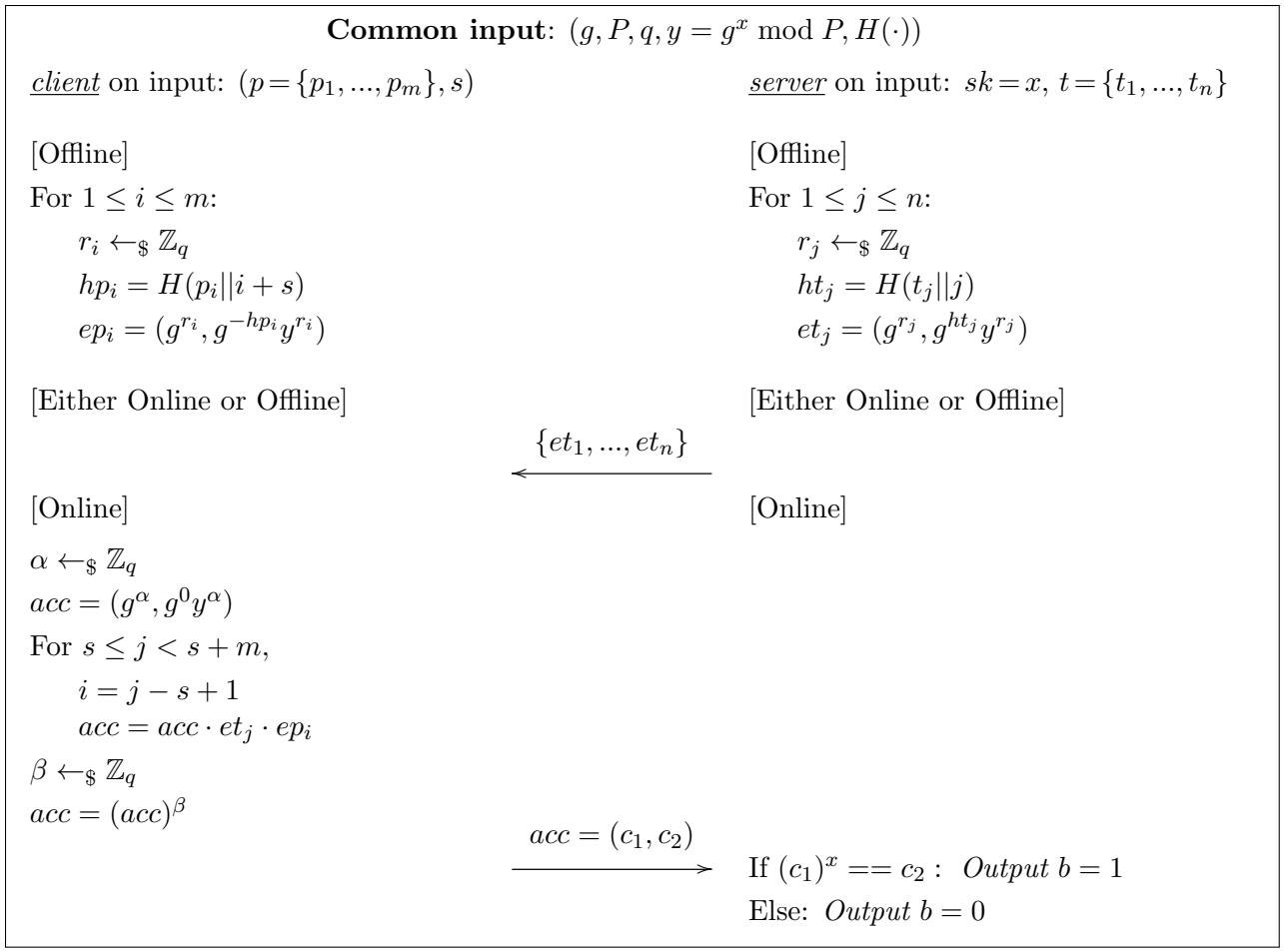


Figure 5.2: AH-ElGamal based SPH-PSM. (Computation is done mod P).

5.1.3 Security Analysis

Theorem. If AddHomEnc is an IND-CPA additively homomorphic cryptosystem, the protocol in Figure 5.1 correctly computes SPH-PSM with both *server*'s and *client*'s privacy.

Proofs. We now show that the protocol satisfies correctness and both parties' privacy, as defined in Section 5.1.3.

Correctness. We show that $b = 1$ if and only if $t[s : m] = p$ except for probability negligible in λ. First, we note that:

$$t[s:m] = p \iff \forall_{i=1,\ldots,m}\ t_{s+i-1} = p_i \Longrightarrow \sum_{i=1}^{m} (t_{s+i-1} - p_i)^{r_i} = 0$$

Furthermore, since:

$$acc = \prod_{i=1}^{m} (E(t_{s+i-1}) \cdot E(-p_i))^{r_i} = E\left(\sum_{i=1}^{m} (t_{s+i-1} - p_i)^{r_i}\right)$$

it follows that:

$$\begin{aligned} t[s:m] = p \Longrightarrow \textstyle\sum_{i=1}^{m} (t_{s+i-1} - p_i)^{r_i} = 0 \quad &\iff acc = E(0) \\ &\iff b = 1 \end{aligned}$$

To complete the proof, we only need to show that:

$$\sum_{i=0}^{m} (t_{s+i-1} - p_i)^{r_i} = 0 \Longrightarrow t[s:m] = p$$

Note that, if p_i does not match t_{s+i}, then the i-th element in the sum is a random group element r'_i.

$$t_{s+i-1} \neq p_i \Longrightarrow (t_{s+i-1} - p_i)^{r_i} = r'_i$$

Thus, the sum comprises m random group elements. However, it equals zero with probability $1/q$, which is negligible in λ. Hence, if the sum is zero, then it must hold that $t[s:m] = p$, with overwhelming probability.

Client's Privacy. To show that *server*'s view can be efficiently simulated, we construct a simulator B^*. On input of (t, b), B^* outputs the real transcript, except for acc – the only value received by *server*– which B^* computes as: $acc = E(0)$ if $b = 1$ and $acc = E(r)$ for $r \leftarrow_{\$} \mathbb{Z}_q$, otherwise. Note that acc is distributed identically to the real execution, since, as noted above, if $b = 0$, then acc is the encrypted sum of one or more random group elements and/or many zeros, and is itself an encryption of a random group element.

Server's Privacy. To show *Server*'s privacy, we construct a simulator A^*. On inputs $((p, s), n)$, A^* first outputs $\{et'_1, ..., et'_n\}$ where $et'_j = \{E(r_j)\}$ and $r_j \leftarrow_\$ \mathbb{Z}_q$. Then, A^* outputs the rest of the view, as usual. Based on indistinguishability of AddHomEnc, it holds that: $\forall j\ et'_j \equiv^c et_j$, hence the view and the simulation are computationally indistinguishable.

5.1.4 Timing Attacks

client's and *server*'s privacy properties discussed above guarantee that each party only learns what it is intended to learn, based on the information (i.e., protocol messages) exchanged. However, they do not take into account auxiliary means of information leakage, such as the timing channel.

It is easy to see that *client*'s computation in the online phase of the protocol is proportional to m – search substring size. Thus, assuming negligible message transmission delay and knowledge of the *client*'s computing platform, *server* (or anyone observing the protocol) could estimate m.[1]

There are several ways to counter such timing attacks. One trivial counter-measure is for *client* to pipeline (or parallelize) transfer of the encrypted genome, i.e., $\{et_1, ..., et_n\}$, with the computation of *acc* in the loop of Figure 5.1. Since $m << n$, $O(m)$ computation overhead at *client* is dominated by the $O(n)$ communication overhead incurred by transfer of the encrypted genome. Though this is easy to achieve, *client* would have to wait to reply with *acc* until all ciphertexts are received, regardless of the starting location of the search substring.

Alternatively, we could consider transfer of the encrypted genome as part of the protocol's offline phase. Then, the online phase begins with *client* performing $O(m)$ operations and no other party can infer m since it is not privy to *client*'s *actual* starting time. This might,

[1]Note that timing attacks against *server* are not meaningful in our context.

in fact, reflect a real-world setting where a testing facility receives encrypted genomes from many patients and batches computation.

Finally, *client* could delay sending the final protocol message containing *acc* for a certain period. If this delay matches the time to compute a pattern of length m', *client* can effectively pad m to m'. With the optimized version of our protocol (see Section 5.2.2), *client*'s $O(m)$ operations reflect inexpensive modular multiplications, which can be completed, for $m \leq 10^8$, in under one minute. Thus, m' can be several orders of magnitude greater than m, without unduly affecting the overall run-time of the protocol.

5.2 SPH-PSM in Practice

We now discuss some practical considerations for implementing SPH-PSM.

5.2.1 Instantiating AddHomEnc

As discussed in Section 5.1.2, SPH-PSM relies on availability of AddHomEnc, i.e., IND-CPA-secure additively homomorphic encryption. There are several suitable candidates, including Paillier [Pai99], Dramgard-Jurik [DJ01], and Okamato-Uchiyama [OU98].

However, it is easy to see that SPH-PSM does not actually require ability to decrypt arbitrary ciphertexts. In fact, for the purposes of SPH-PSM, it suffices to test whether a ciphertext is an encryption of zero. This allows us to consider the Additively Homomorphic ElGamal variant (AH-ElGamal) as one of the choices.

AH-ElGamal involves the following algorithms:

- *Key Generation:* On input of security parameter λ, select public parameters (g, P, q) for primes P, q and g generator of subgroup of $\mathbb{Z}_P$ of order q. (Note: we use P instead of more customary p, since, in our notation, p denotes *client*'s substring). Private key SK is $x \leftarrow_{\$} \mathbb{Z}_q$ and public key PK is $y = g^x \bmod P$.

- *Encryption:* $E_{PK}(m) = (c_1, c_2)$, where $c_1 = g^r \bmod P$ and $c_2 = g^m y^r \bmod P$, for $r \leftarrow_{\$} \mathbb{Z}_q$.

- *Testing for Encryption of Zero:* $D_{SK}(c_1, c_2) = 0$ if and only if $c_2 = (c_1)^x \bmod P$.

There is also an Elliptic Curve-based ElGamal variant (EC-ElGamal) that is additively homomorphic [UWL+09]. It involves the following algorithms:

- *Key Generation:* On input of security parameter λ, select public parameters (P, e, G, η), where e is an elliptic curve over finite field $\mathbb{GF}(P)$, η is the order of curve e, and G is the generator point of e. SK is integer $x \in \mathbb{GF}(P)$, and PK is $Y = xG$.

- *Encryption:* $E_{PK}(m) = (R, S)$, where $R = rG$ and $S = M + rY$, for $r \in [1, \eta - 1]$.

- *Decryption:* $D_{SK}(R, S) = -xR + S$.

Naturally, to efficiently realize SPH-PSM, we need an AddHomEnc instantiation that meets the following criteria: (1) ability to test for encryption of zero, (2) small ciphertext size, (3) fast encryption, and (4) fast homomorphic addition operations.

Actually, the "weight" of such factors depends on whether transfer of *server*'s encrypted genome is part of the offline phase. If so, complexity of SPH-PSM is clearly dominated by *client*'s $O(m)$ computation overhead. Thus, the preferred instantiation is AH-ElGamal due to its computational efficiency. Since random exponents can be taken from a subgroup (e.g., 160-bit), encryptions and re-randomizations can be computed efficiently, e.g., relative to Paillier.

	AH-ElGamal			EC-ElGamal		
m = |p|	**Online**	**Online Optimized**	**Substring Prepr.**	**Online**	**Online Optimized**	**Substring Prepr.**
10	0.76ms	0.1ms	0.46ms	12.71ms	0.78ms	7.05ms
10^2	4.51ms	0.22ms	4.05ms	66.15ms	3.77ms	36.9ms
10^3	28.72ms	0.68ms	20.35ms	635.99ms	32.64ms	317.62ms
10^4	291.12ms	6.02ms	168.28ms	6.41s	318.65ms	3.17s
10^5	2.86s	60.18ms	1.63s	1m 7s	3.28s	31.95s
10^6	28.61s	647.05ms	16.9s	10m 35s	31.85s	5m 17s
10^7	4m 45s	6.25s	2m 41s	1h 45m	5m 19s	52m 55s
10^8	47m 37s	1m 2s	26m 53s	17h 38m.	53m 17s	8h 49m
10^9	7h 56m	10m 25s	4h 28m	7d 8h	8h 52m	3d 16h
$3 \cdot 10^9$	23h 48m	31m 16s	13h 26m	22d 1h	1d 2h	11d 54s

Table 5.2: SPH-PSM: computation time for varying substring lengths.

Conversely, if transferring the encrypted genome is part of the online phase, transfer dominates complexity of SPH-PSM. Thus, the preferred instantiation is EC-ElGamal, since group elements are much smaller (160-bit), as opposed to 1024-bit in AH-ElGamal and 2048-bit in Paillier, using the same security parameters.

5.2.2 ElGamal-based SPH-PSM

We now present an ElGamal-based instantiation of SPH-PSM and introduce some optimizations. We focus on reducing computational complexity of the online phase. Although we describe this instantiation in terms of AH-ElGamal, the discussion applies to EC-ElGamal as well.

In AH-ElGamal, multiplying two ciphertexts (adding two corresponding plaintexts) is markedly more efficient than exponentiating a ciphertext with a constant (multiplying the corresponding plaintext by the constant). Therefore, we modify the basic protocol of Figure 5.1 such that *client*'s online phase only involves multiplications. Rather than encrypting a raw nucleotide, *server* encrypts the hash of the nucleotide along with its corresponding position in the genome. To this end, we need a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ (modeled as a random oracle)[2].

[2]Without a hash function, there can be cross cancellation between positions.

The resulting protocol is shown in Figure 5.2. *client* performs $O(m)$ modular multiplications and only $O(1)$ modular exponentiations during the online phase. This greatly reduced online overhead also helps thwart timing attacks discussed in Section 5.1.4.

5.2.3 Performance Evaluation

Protocols discussed in this chapter have been implemented in `C++`, and tested on a desktop machine running Ubuntu 12.10, with an Intel i7-3770 3.4GHz quad-core CPU and 16GB of RAM. We implemented 1024-bit AH-ElGamal using the gmp [gmp16] library and 160-bit EC-ElGamal – using the OpenSSL library [Ope16]. We break up the measurements as follows:

1. Genome Encryption – time for *Server* to encrypt his genome.
2. Data Transfer – time for *client* to download *server*'s genome.
3. Substring Pre-processing – time for *Client* to pre-process her substring.
4. Online Phase – time elapsed after genome is received, until *server* outputs the result, i.e., b. (This phase is dominated by *client*'s computation).

Experiments were conducted with *client*'s substring size ranging from 10 to $3 \cdot 10^9$ letters. For tests taking longer than several hours, run-times have been estimated by performing tests on a "smaller" genome (10^7 nucleotides). This is justifiable since tested protocols incur linear complexities.

We also note that our experiments were conducted rather conservatively, using a single thread on a regular (mid-range) desktop machine. More realistic scenarios would involve machines with multiple cores, much bigger RAM and higher-end CPU-s.

First, we measured genome encryption: it took approximately 115 hours using AH-ElGamal and an impressive $2,580$ hours using EC-ElGamal. This significant difference can be explained by the fact that AH-ElGamal is implemented using gmp, whereas, EC-ElGamal – using OpenSSL, which is markedly slower. While we do plan to optimize EC-ElGamal performance, this issue is somewhat secondary, since: (1) EC-ElGamal is used in settings where first priority is to minimize communication overhead, and (2) a genome is encrypted only once, ahead of time, and (3) encryption can be easily parallelized, using multiple cores.

Next, we estimate the time to transfer the encrypted genome. On a 1Gbps link, it takes approximately 2.7 hours with EC-ElGamal and 8.7 hours with AH-ElGamal. Due to linear complexity of SPH-PSM, transfer times over networks with different speed can be straightforwardly inferred, e.g., 27 hours with EC-ElGamal and 87 hours with AH-ElGamal over a 100Mbps link.

Finally, Table 5.2 reports on run-times for *client*'s substring pre-processing and for the online phase dominated by *client*'s computation. For the latter, we quantify the speedup due to the optimization presented in Section 5.2.2.

Although we report on tests conducted up to the largest possible substring, i.e., the entire genome ($3 \cdot 10^9$ letters), most genomic applications used today require testing of significantly smaller substrings. For example, prior work in [BBD+11] and [DCFGT12] supported personalized medicine testing with fewer than $1,002$ markers, e.g., analysis of the *tpmt* gene common in leukemia patients, or testing for *hla-b* variants associated to sensitivity to some HIV drugs. For this class of tests, our optimized SPH-PSM protocol completes in less than minute. However, it is still feasible to securely conduct future tests with much larger substrings.

5.3 Extensions

We now discuss some natural extensions to the basic SPH-PSM protocol.

5.3.1 Revealing Test Outcome to Alice

Under some circumstances, it might be necessary for both *server* and *client* to learn the test outcome. For example, "*client*" might actually be a medical laboratory and it could be in the patient's best interest to reveal the test outcome to a specialist as soon as possible.

In the HbC model,we can trivially extend the basic SPH-PSM protocol to allow *server* to communicate b to *client*. Nonetheless, we leave it for future work to explore the use of threshold cryptosystems to provably enforce mutual output in SPH-PSM, e.g., by relying on threshold ElGamal [DF89] or threshold Paillier [FP01, DJ01].

5.3.2 Fixed-Size Wildcards and Non-Contiguous Substrings

client's substring might contain fixed-length wildcards or it might be non-contiguous. There is no real difference between a substring such as "ACG??TAAC" (where "?" is a single-character wildcard) and a two-part (non-contiguous) substring "ACG" and "TAAC", with each part starting at some locations s_1 and s_2, that are arbitrary number of positions apart, e.g., $s_2 = s_1 + 5$.

It is easy to see that our basic protocol is independent of contiguity of *Client*'s substring. Computational complexity remains $O(m)$.

5.3.3 Multiple Substrings/Starting Locations

In some applications, *client* might not know the exact starting position of a particular search substring: it could begin in several (v) possible positions. This situation is typical in genomic testing since alignment of the patient's genome may not always be exactly precise. We can easily extend SPH-PSM such that *client* only learns whether ***any*** one of these positions match. The resulting extension is a trivial variation of the basic version. The resulting protocol only incurs $O(m * v)$ computation overhead, at *client*'s side, as compared to $O(m)$ in the basic protocol; communication and *server*'s computation overheads are unaltered.

A similar case occurs when, even if the exact starting position is known, a particular position in a search substring can be ***one of several choices***. For example, "GA[G,T]TACA" denotes that position 3 in the substring can be either G or T. This is clearly distinct from a single-character wildcard, since neither A nor C is allowed in this position. The natural way to deal with such disjunctive conditions is to try to match two substrings: "GATTACA" and "GAGTACA" starting at the same position. The very same protocol extension applies here.

Yet another variation occurs whenever multiple distinct substrings $(p_1, \ldots, p_w)$ are matched in parallel or disjunctively. In this case *server* learns ***how many*** matches occur. This case is slightly different from that of non-contiguous substrings or wildcards mentioned above, where *server* only learns the result of a single test. Again, the same extension handles this case. Communication and *server*'s computation complexities are unaltered, while *client*'s computation overhead would amount to $O(\sum_{i=1}^{w} |p_i|)$, as compared to $O(|p| = m)$ in the basic version.

5.3.4 Reducing Data Transfer Time

The basic protocol entails *server* sending the entire letter-by-letter encrypted genome to *client*. This constitutes the most costly part of the protocol – about $3.2 \cdot 10^9$ ciphertexts. With EC-ElGamal, where each ciphertexts is a 320-bit value, transferring the encrypted genome translates into roughly 100GB of data.

However, this massive data transfer needs to happen only once for many tests that *server* (e.g., a given patient) conducts with *client* (e.g., a particular medical facility). Also, while transmitting encrypted genomes in real time might be unfathomable today, it will likely become realistic in not-so-distant future. Plus, most genomic tests are not typically spontaneous, i.e., patients plan and schedule them in advance. Consequently, transfer of the encrypted genome can take place incrementally over some period preceding the actual test. In fact, a testing facility could interact with many patients at a time, gradually uploading their encrypted genomes. Once a given patient's genome is fully uploaded, the actual protocol begins.

Physical Transfer. For some tests, *server* might physically visit the testing facility (*client*). In that setting, *server* might be asked to bring along a dedicated device, e.g., a USB stick or a portable flash drive, containing the entire encrypted genome. Such passive devices capable of storing over 1TB of data are readily available today and cost well below US$100. Once plugged into *client*'s computing equipment, much faster transfer rates apply: for example, USB 3.0 offers transfer rates of up to 4.8Gbps, which translates into about 3 minutes for transferring the encrypted genome.

Alternatively, *client* could simply read the desired m ciphertexts directly from *server*'s drive and perform computation on them, without bothering with any other data. This would clearly minimize communication delay. However, care must be taken to prevent attacks that

might exploit the portable drive's logs or other data to later infer locations that have been read by *client*.

Leveraging Cloud Storage. Another possibility is to use the cloud. By storing *server*'s encrypted genome at a cloud provider, transfer of the encrypted genome would need to be performed only once, for all possible tests and testing facilities. The cloud provider can offer both higher bandwidth and availability. However, if the cloud provider is also trusted not to collude with *server*, *client* could avoid bulk transfer and selectively fetch only desired ciphertexts. (A collusion would reveal the size and position of the substring held by *client*.)

One potential concern, even without any collusion, is that the cloud provider would learn the size and position of *client*'s substring. Fortunately, the latter can be mitigated by letting *client* and *server* pre-agree on a common secret that *server* could later use to shuffle encrypted nucleotides before uploading to the cloud. This way, ordering of ciphertexts (but not m – the substring size) would be obscured from the cloud provider.

Alternatively, *client* could rely on cryptographic techniques for Private Information Retrieval (PIR) [CGKS95] to securely retrieve m desired ciphertexts from the cloud. Despite PIR's computational overhead, recent results [BPMO12, MBC13] show that PIR can be efficiently adapted to cloud settings where a static database is maintained in a MapReduce cluster. However, the size of *client*'s substring would be revealed to the cloud provider.

5.3.5 Coping with (Some) Malicious Input

We now consider the case where malicious *server* tries to guess the specifics of *client*'s test, by verifying its guess of *client*'s input. Let (p, s) denote *client*'s input and (p', s') – *server*'s guessed version. While faithfully following all SPH-PSM protocol steps, *server* inputs a concocted genome t' where $t'[s' : m'] = p'$ and $m' = |p'|$. In other words, the substring

in t' starting at position p' is set to s'. (We assume, without loss of generality, that s and s' are contiguous substrings). For all other positions, *server* sets $t_j = X$ for some $X \notin \{A, C, G, T, -\}$. Then, *server* uses t' as input for SPH-PSM with *client*. It is easy to see that, if *acc* decrypts to zero, *server* learns that $(p, s) = (p', s')$.

On one hand, in the HbC model, *server* is assumed not to deviate from the protocol, which should rule out this guessing attack. On the other hand, while HbC is clear with respect to participants scrupulously following all protocol steps, it is somewhat murky with regard to the validity or goodness of participants' input. To err on the side of caution, we can address *server*'s input validity (and hence guessing attacks such as the one above) by indirectly integrating the genome sequencing lab into the domain of SPH-PSM.

Specifically, at sequencing time, the genome sequencing facility (denoted by Charlie) could offer some additional services beyond supplying a digitized genome:

- Encrypt the genome, one nucleotide at a time, under *server*'s public key to produce $et_1, ..., et_n$. This is a reasonable service for Charlie to provide, since it anyhow learns *server*'s genome and must communicate it back securely.

- Sign the encrypted genome $(et_1, ..., et_n)$ as a whole. Similarly, to prevent any kind of future disputes and provide overall integrity as well as origin authenticity, it makes sense for Charlie to sign what it produces.

Armed with these modifications we can easily amend SPH-PSM to let *server* send *client* a *signed* encrypted genome. By verifying Charlie's signature, *client* can be assured that *server*'s input is a valid genome sequenced by a reputable entity – Charlie. This change has negligible effect on performance if transfer of the encrypted genome is part of the online phase. This is because *client* needs to anyway receive all ciphertexts; it can gradually hash the encrypted genome as it is being received and, at the end, verify Charlie's signature.

However, if transfer is done off-line (as in Section 5.3.4) or the encrypted genome is accessed piece-meal from a cloud provider, signing the entire encrypted genome is not useful. In this case – also at sequencing time – Charlie could individually sign each ciphertext (encrypted nucleotide). During the off-line phase, *client* can obtain selected ciphertext directly (from the cloud provider or from *server*'s flash drive or USB stick) and easily verify that each et_j, for $s \leq j < s + m$, is accompanied by a valid Charlie's signature.

Chapter 6

Efficient Pattern Matching on Symetrically Encrypted Data

This chapter addresses the problem of private pattern matching over large amounts of symetrically encrypted data. It is based on my publication It is based on my publication *Rich Queries on Encrypted Data: Beyond Exact Matches* published in the European Symposium on Research in Computer Security written along with Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel Rosu, and Michael Steiner [FJK+15]. While the presentation has changed no extentions have been made.

Searchable symmetric encryption (SSE) addresses a setting where a client outsources an encrypted database (or document/file collection) to a remote server $\mathcal{E}$ such that the client, which only stores a cryptographic key, can later search the collection at $\mathcal{E}$ while hiding information about the database and queries from $\mathcal{E}$. Leakage to $\mathcal{E}$ is to be confined to well-defined forms of data-access and query patterns while preventing disclosure of explicit data and query plaintext values. SSE has been extensively studied [SWP00, Goh03, CM05, CGKO06, CK10, vLSD+10, KPR12, KO12, KP13, CJJ+13, JJK+13, CJJ+14, PVK+14,

NPG14], particularly in last years due to the popularity of clouds and data outsourcing, focusing almost exclusively on single-keyword search.

Recently, Cash et al. [CJJ+13] and Pappas et al. [PVK+14] presented the first SSE solutions that go well beyond single-keyword search by supporting Boolean queries on multiple keywords in sublinear time. In particular, [CJJ+13, CJJ+14] build a very scalable system with demonstrated practical performance with databases containing indexes in the order of tens of billions document-keyword pairs. In this chapter we extend the search capabilities of the system from [CJJ+13] (*referred to as the OXT protocol*) by supporting substring queries (e.g., return records with textual information containing a given pattern, say 'crypt'), wildcard queries (combining substrings with one or more single-character wildcards), and phrase queries (return records that contain the phrase "searchable encryption"). Moreover, by preserving the overall system design and optimized data structures of [CJJ+14], we can run any of these new queries in combination with Boolean-search capabilities (e.g., combining a range and/or substring query with a conjunction of additional keywords/ranges/substrings) and we can do so while preserving the scalability of the system and additional properties such as support for *dynamic data*.

We also show how to extend our techniques to the more involved multi-client SSE scenarios studied by Jarecki et al. [JJK+13]. In the first scenario, denoted MC-SSE, the owner of the data, $\mathcal{D}$, outsources its data to a remote server $\mathcal{E}$ in encrypted form and later allows multiple clients to access the data via search queries and according to an authorization policy managed by $\mathcal{D}$. The system is intended to limit the information learned by clients beyond the result sets returned by authorized queries while also limiting information leakage to server $\mathcal{E}$. A second scenario, OSPIR-SSE or just OSPIR (for Outsourced Symmetric PIR), addresses the multi-client setting but adds a requirement that $\mathcal{D}$ can authorize queries to clients following a given policy, but without $\mathcal{D}$ learning the specific values being queried. That is, $\mathcal{D}$ learns

minimal information needed to enforce policy, e.g., the query type or the field to which the keyword belongs, say `last name`, but not the actual last name being searched.

The queries we support, i.e. substrings, wildcards and phrases, are all derived from a novel technique that allows us to search on the basis of positioning information (where the data and the position information are encrypted). This technique can be used to implement any query type that can be reduced to Boolean formulas on queries of the form "are two data elements at distance Δ?". For example, in the case of substring queries, the substring is tokenized (i.e., subdivided) into a sequence of possibly-overlapping k-grams (strings of k characters) and the search is performed as a conjunction of such k-grams. However, to avoid false positives, i.e., returning documents where the k-grams appear but not at the right distances from each other, we use the relative positions of the tokens to ensure that the combined k-grams represent the searched substring. Wildcard queries are processed similarly, because t consecutive wildcard positions (i.e., positions that can be occupied by any character) can be implemented by setting the distance between the two k-grams that bracket the string of t wildcards to $k + t$. Phrase queries are handled similarly, by storing whole words together with their encrypted positions in the text.

The crux of this technique is a homomorphic computation on encrypted position information that gives rise to a very efficient SSE protocol between client $\mathcal{C}$ and server $\mathcal{E}$ for computing relative distances between data elements while concealing this information from $\mathcal{E}$. This protocol meshes naturally with the homomorphic properties of OXT but in its general form it requires an additional round of interaction between client and server. In the SSE setting, the resulting protocol preserves most of the excellent performance of the OXT protocol (with the extra round incurring a moderate increase in query processing latency). For the OSPIR setting we resort to bilinear groups for some homomorphic operations, hence impacting performance in a more noticeable way which we are currently investigating.

We prove the security of our protocols in the SSE model of [CGKO06, CK10, CJJ+13], and the extensions to the MC-SSE and OSPIR settings of [JJK+13], where security is defined in the real-vs-ideal model and is parametrized by a specified leakage function $\mathcal{L}(\mathsf{DB}, \mathbf{q})$. A protocol is said to be secure with leakage profile $\mathcal{L}(\mathsf{DB}, \mathbf{q})$ against adversary $\mathcal{A}$ if the actions of $\mathcal{A}$ on adversarially-chosen input DB and query set $\mathbf{q}$ can be simulated with access to the leakage information $\mathcal{L}(\mathsf{DB}, \mathbf{q})$ only (and not to DB or $\mathbf{q}$). This allows modeling and bounding the partial leakage incurred by SSE protocols. It means that even an adversary that has full information about the database and queries, or even chooses them at will, does not learn anything from the protocol execution other than what can be derived solely from the defined leakage profile. We achieve provable *adaptive* security against adversarial servers $\mathcal{E}$ and $\mathcal{D}$, and against malicious clients. Servers $\mathcal{E}$ and $\mathcal{D}$ are assumed to return correct results (e.g., server $\mathcal{E}$ returns all documents specified by the protocol) but can otherwise behave maliciously. However, in the OSPIR setting, query privacy from $\mathcal{D}$ is achieved as long as $\mathcal{D}$ does not collude with $\mathcal{E}$.

Practicality of our techniques was validated by a comprehensive implementation of the SSE protocols for substring and wildcard queries, and their combination with Boolean functions on exact keywords. These implementations (extending those of [CJJ+13, JJK+13, CJJ+14]) were tested by an independent evaluator on DB's of varying size, up to 10 Terabytes with 100 million records and 25.6 billion record-keyword pairs. Performance was compared to MariaDB's (an open-source fork of MySQL) performance on the same databases running on *plaintext data and plaintext queries*. Due to the highly optimized protocols and careful I/O management, the performance of our protocols matched and often exceeded the performance of the plaintext system. These results are presented in Section 6.4.

Related Work. The only work we are aware of that addresses substring search on symmetrically encrypted data is the work of Chase and Shen [CS14]. Their method, based on suffix trees, is very different than ours and the leakage profiles seem incomparable. This is

a promising direction, although the applicability to (sublinear) search on large databases, and the integration with other query types, needs to be investigated. Its potential generalization to the multi-client or OSPIR settings is another interesting open question. Range and Boolean queries are supported, also for the OSPIR setting, by Pappas et al. [PVK+14] (building on the work of Raykova et al [RVBM09]). Their design is similar to ours in reducing range queries to disjunctions (with similar data expansion cost) but their techniques are very different offering an alternative (and incomparable) leakage profile for the parties. The main advantages of our system are the support of the additional query types presented here and its scalability. The scalability of [PVK+14] is limited by their crucial reliance on Bloom filters that requires database sizes whose resultant Bloom filters can fit in RAM. A technique that has been suggested for resolving range queries in the SSE setting is *order-preserving encryption* (e.g., it is used in the CryptDB system [PRZB11]). However, it carries a significant intrinsic loss of privacy as the ordering of ciphertexts is visible to the holding server (and the encryption is deterministic). Range queries are supported in the multi-writer public key setting by Boneh-Waters [BW07] and Shi et al. [SBC+07] but at a significantly higher computational cost.

6.1 Preliminaries

Our work concerns itself with databases in a very general sense, including relational databases (with data arranged in "rows" and "columns"), document collections, textual data, etc. We use interchangeably the word 'document' and 'record'. We think of keywords as (attribute,value) pairs. The attribute can be structured data, such as name, age, SSN, etc., or it can refer to a textual field. We sometimes refer explicitly to the keyword's attribute but most of the time it remains implicit. We denote by m the number of distinct attributes and use $I(w)$ to denote the attribute of keyword w.

SSE protocols and formal setting (following [CJJ+13]). Let τ be a security parameter. A database $\mathsf{DB} = (\mathsf{ind}_i, \mathsf{W}_i)_{i=1}^d$ is a list of identifier and keyword-set pairs, where $\mathsf{ind}_i \in \{0,1\}^\tau$ is a document identifier and $\mathsf{W}_i = \mathsf{DB}[\mathsf{ind}_i]$ is a list of its keywords. Let $\mathsf{W} = \bigcup_{i=1}^d \mathsf{W}_i$. A *query* PSI is a predicate on W_i where $\mathsf{DB}(\mathsf{PSI})$ is the set of identifiers of document that satisfy PSI. E.g. for a single-keyword query we have $\mathsf{DB}(w) = \{\mathsf{ind} \text{ s.t. } w \in \mathsf{DB}[\mathsf{ind}]\}$.

A *searchable symmetric encryption (SSE) scheme* Π consists of an algorithm Setup and a protocol Search fitting the following syntax. Setup takes as input a database DB and a list of document (or record) decryption keys RDK, and outputs a secret key K along with an encrypted database EDB. The search protocol Search proceeds between a *client* $\mathcal{C}$ and *server* $\mathcal{E}$, where $\mathcal{C}$ takes as input the secret key K and a query PSI and $\mathcal{E}$ takes as input EDB. At the end of the protocol, $\mathcal{C}$ outputs a set of $(\mathsf{ind}, \mathsf{rdk})$ pairs while $\mathcal{E}$ has no output. We say that an SSE scheme is *correct* for a family of queries Ψ if for all $\mathsf{DB}, \mathsf{RDK}$ and all queries $\mathsf{PSI} \in \Psi$, for $(K, \mathsf{EDB}) \leftarrow \mathsf{Setup}(\mathsf{DB}, \mathsf{RDK})$, after running Search with client input (K, PSI) and server input EDB, the client outputs $\mathsf{DB}(\mathsf{PSI})$ and $\mathsf{RDK}[\mathsf{DB}(\mathsf{PSI})]$ where $\mathsf{RDK}[S]$ denotes $\{\mathsf{RDK}[\mathsf{ind}] \,|\, \mathsf{ind} \in S\}$. Correctness can be statistical (allowing a negligible probability of error) or computational (ensured only against computationally bounded attackers - see [CJJ+13]).

Note (retrieval of matching encrypted records). Above we define the output of the SSE protocol as the set of identifiers ind pointing to the encrypted documents matching the query (together with the set of associated record decryption keys rdk). The retrieval of the document payloads, which can be done in a variety of ways, is thus decoupled from the storage and processing of the metadata which is the focus of the SSE protocols.

Multi-Client SSE Setting [JJK+13]. The MC-SSE formalism extends the SSE syntax by an algorithm $\mathsf{GenToken}$, which generates a search-enabling value token from the secret key K generated by the data owner $\mathcal{D}$ in Setup, and query PSI submitted by client $\mathcal{C}$. Protocol

Search is then executed between server $\mathcal{E}$ and client $\mathcal{C}$ on resp. inputs EDB and token, and the protocol must assure that $\mathcal{C}$ outputs sets DB(PSI) and RDK[DB(PSI)].

OSPIR SSE Setting [JJK+13]. An OSPIR-SSE scheme replaces the GenToken procedure, which in MC-SSE is executed by the data owner $\mathcal{D}$ on the cleartext client's query q, with a two-party protocol between $\mathcal{C}$ and $\mathcal{D}$ that allows $\mathcal{C}$ to compute the search-enabling token without $\mathcal{D}$ learning PSI. However, $\mathcal{D}$ should be able to enforce a query-authorization policy on $\mathcal{C}$'s query. We consider attribute-based policies, where queries are authorized based on the attributes associated to keywords in the query (e.g., a client may be authorized to run a range query on attribute 'age' but not on 'income', or perform a substring query on the 'address' field but not on the 'name' field, etc.). Later, we will consider extensions where the policy can define further constraints, e.g., the total size of an allowed interval in a range query, or the minimal size of a pattern in a substring query. An attribute-based policy for any query type is represented by a set of attribute-sequences P s.t. a query PSI involving keywords (or substrings, ranges, etc) $(w_1, ..., w_n)$ is *allowed by policy* P if and only if the sequence of attributes $\mathsf{av}(\mathsf{PSI}) = (I(w_1), ..., I(w_n)) \in \mathsf{P}$. Using this notation, the goal of the GenToken protocol is to let $\mathcal{C}$ compute token corresponding to its query on PSI only if $\mathsf{av}(\bar{w}) \in \mathsf{P}$. Note that different query types will have different entries in P. Reflecting these goals, an OSPIR-SSE scheme is a tuple $\Sigma = (\mathsf{Setup}, \mathsf{GenToken}, \mathsf{Search})$ where Setup and Search are as in MC-SSE, but GenToken is a protocol run by $\mathcal{C}$ on input PSI and by $\mathcal{D}$ on input (P, K), with $\mathcal{C}$ outputting token if $\mathsf{av}(\mathsf{PSI}) \in \mathsf{P}$, or $\perp$ otherwise, and $\mathcal{D}$ outputting $\mathsf{av}(\mathsf{PSI})$.

6.2 Substring Queries

Our substring-search capable SSE scheme is based on the conjunctive-search SSE protocol OXT of [CJJ+13], and it extends that protocol as follows: Whereas the OXT scheme of

[CJJ+13] supported efficient retrieval of records containing several required keywords at once (i.e. satisfying a *conjunction* of several keyword-equality search terms), our extension supports efficient retrieval of records containing the required keywords *at required relative positions to one another*. This extension of conjunctive search with positional distance criteria allows us to handle several query types common in text-based information retrieval. To simplify the description, and using the notation from Section 6.1, consider a database $\mathsf{DB} = (\mathsf{ind}_i, T_i)$ containing records with just one free text attribute, i.e. where each record T_i is a text string. We support the following types of queries q:

Substring Query. Here q is a text string, and $\mathsf{DB}(q)$ returns all ind_i s.t. T_i contains q as a substring.

Wildcard Query. Here q is a text string which can contain wildcard characters $'?'$ (matching any single character), and $\mathsf{DB}(q)$ returns all ind_i s.t. T_i contains a substring q' s.t. for all j from 1 to $|q|$, $q_j =' ?' \ \vee \ q_j = q'_j$, where q_j and q'_j denote j-th characters in strings q and q'. If the query should match only prefixes (suffixes) of T_i, the query can be prefixed (suffixed) with a $'\hat{}'$ ($'\$'$).

Phrase Query. Here q is a sequence of words, i.e. text strings, $q = (q^1, \ldots, q^l)$, where each q^i can equal to a wildcard character $'?'$. Records T_i in DB are also represented as sequences of words, $T_i = (T_i^1, \ldots, T_i^n)$. $\mathsf{DB}(q)$ returns all ind_i s.t. for some k and for all j from 1 to l, it holds that $q^j =' ?' \vee q^j = T_i^{k+j}$. (Note that phrase queries allow a match of a single wildcard with a whole word of any size, while in a wildcard query a single wildcard can match only a single character.)

All these query types utilize the same crypto machinery that we describe next for the substring case. In Section 6.2.2 we explain briefly how to adapt the techniques to these queries too.

6.2.1 Basic SSE Substring Search

Here we present protocol SUB-SSE-OXT that supports substring search in the basic SSE model (i.e., a single client $\mathcal{C}$ outsources its encrypted database to server $\mathcal{E}$) and where the query consists of a single substring. This simpler case allows us to explain and highlight the basic ideas that we also use for addressing the general case of boolean expressions that admit substrings as the expression terms as well as for extending these solutions to the more involved MC and OSPIR settings.

Figure 6.1 describes the protocol where shadowed text highlights the changes with respect to the original OXT protocol from [CJJ+13] for resolving conjunctive queries in the SSE model (the reader can visualize the underlying OXT protocol by omitting the shadowed text). We first explain the basic rationale and functioning of the conjunctive-search OXT protocol, and then we explain how we extend it by imposing additional constraints on *relative positions* of the searched terms, and how this translates into support for substring-search SSE.

The Conjunctive SSE Scheme OXT. Let $q = (w_1, \ldots, w_n)$ be a conjunctive query where $\mathsf{DB}(q) = \cap_{i=1}^{n} \mathsf{DB}(w_i)$. Let F_G be a Pseudorandom Function (PRF) with key K_G. (This PRF will map onto a cyclic group G, hence the name.) Let the setup algorithm create as metadata a set of (keyed) hashes XSet, named for "cross-check set", containg the hash values $\mathsf{xtag}_{w,\mathsf{ind}} = F_G(K_G, (w, \mathsf{ind}))$ for all keywords $w \in \mathsf{W}$ and records $\mathsf{ind} \in \mathsf{DB}(w)$. Let the setup also create the matadata needed to quickly retrieve the set of record indexes $\mathsf{DB}(w)$ matching any given *single* keyword $w \in \mathsf{W}$. The OXT protocol is based on a simple conjunctive *plaintext* search algorithm which identifies all records corresponding to a conjunctive query $q = (w_1, \ldots, w_n)$ as follows: It first identifies the set of indexes $\mathsf{DB}(w_1)$ satisfying the first term w_1, called an *s-term*, and then for each $\mathsf{ind} \in \mathsf{DB}(w_1)$ it returns ind as part of $\mathsf{DB}(q)$ if and only if hash value $\mathsf{xtag}_{w_i,\mathsf{ind}} = F_G(K_G, (w_i, \mathsf{ind}))$ is in XSet for all *x-terms* (i.e. "cross-check terms") $w_2, \ldots, w_n$. If group G is sufficiently large then except for negligible collision

probability, if $\mathsf{xtag}_{w_i,\mathsf{ind}} \in \mathsf{XSet}$ for $i \geq 2$ then $\mathsf{ind} \in \cap_{i=2}^{n} \mathsf{DB}(w_i)$, and since ind was taken from $\mathsf{DB}(w_1)$ it follows that $\mathsf{ind} \in \mathsf{DB}(q)$. Since this algorithm runs in $O(|\mathsf{DB}(w_1)|)$ time w_1 should be chosen as the least frequent keyword in q.

To implement the above protocol over *encrypted* data the OXT protocol modifies it in three ways: First, the metadata supporting retrieval of $\mathsf{DB}(w)$ is implemented using single-keyword SSE techniques, specifically the *Oblivious Storage* data structure TSet [CJJ+13, CJJ+14], named for "tuples set", which reveals to server $\mathcal{E}$ only the total number of keyword occurrences in the database, $\sum_{w \in \mathsf{W}} |\mathsf{DB}(w)|$, but hides all other information about individual sets $\mathsf{DB}(w)$ except those actually retrieved during search. (A TSet can be implemented very efficiently as a hash table using PRF F whose key K_T is held by client $\mathcal{C}$, see [CJJ+13, CJJ+14].) Secondly, the information stored for each w in the TSet datastructure, denoted $\mathsf{TSet}(w)$, which $\mathcal{E}$ can recover from TSet given $F(K_T, w)$, is not the plaintext set of indexes $\mathsf{DB}(w)$ but the encrypted version of these indexes using a special-purpose encryption. Namely, a tuple corresponding to the c-th index ind_c in $\mathsf{DB}(w)$ (arbitrarily ordered) contains value $y_c = F_p(K_I, \mathsf{ind}_c) \cdot F_p(K_z, c)^{-1}$, an element in a prime-order group $\mathbb{Z}_p$ where F_p is a PRF onto $\mathbb{Z}_p$, and K_I, K_z are two PRF keys where K_I is global and K_z is specific to keyword w (derived e.g. via another PRF on input w). This encryption enables fast secure computation of hash $\mathsf{xtag}_{w_i,\mathsf{ind}_c}$ between client $\mathcal{C}$ and server $\mathcal{E}$, where $\mathcal{E}$ holds ciphertext $y_c = F_p(K_I, \mathsf{ind}_c) \cdot F_p(K_z, c)^{-1}$ of c-th index ind_c taken from $\mathsf{TSet}(w_1)$ and $\mathcal{C}$ holds keyword w_i and keys K_I, K_z. Let $F_G(K_G, (w, \mathsf{ind})) = g^{F_p(K_X, w) \cdot F_p(K_I, \mathsf{ind})}$ where g generates group G and $K_G = (K_X, K_I)$ where K_X is a PRF key. $\mathcal{C}$ then sends to $\mathcal{E}$:

$$\mathsf{xtoken}[c, i] = g^{F_p(K_X, w_i) \cdot F_p(K_z, c)}$$

for $i = 2, \ldots, h$ and $c = 1, \ldots, |\mathsf{TSet}(w_1)|$, and $\mathcal{E}$ computes $F_G(K_G, (w_i, \mathsf{ind}_c))$ for each c, i as:

$$(\mathsf{xtoken}[c, i])^{y_c} = (\mathsf{xtoken}[c, i])^{F_p(K_I, \mathsf{ind}_c) \cdot F_p(K_z, c)^{-1}}$$

Since K_z is specific to w_1 mask $z_c = F_p(K_z, c)$ applied to ind_c in y_c is a one-time pad, hence this protocol reveals only the intended values $F_G(K_G, (w_i, \mathsf{ind}_c))$ for all $\mathsf{ind}_c \in \mathsf{DB}(w_1)$ and $w_2, \ldots, w_n$.

Extending OXT to Substring SSE. The basic idea for supporting substring search is first to represent a substring query as a conjunction of k-grams (strings of length k) at given relative distances from each other (e.g., a substring query *'yptosys'* can be represented as a conjunction of a 3-gram *'tos'* and 3-grams *'ypt'* and *'sys'* at relative distances -2 and 2 from the first 3-gram, respectively), and then to extend the conjunctive search protocol OXT of [CJJ+13] so that it verifies not only whether the conjunctive terms all occur within the same document, but also that they occur at positions whose relative distances are specified by the query terms. We call representation of a substring q as a set of k-grams with relative distances a *tokenization* of q. We denote the *tokenizer* algorithm as T, and we denote its results as $T(q) = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$ where Δ_i are any non-zero integer values, including negatives, e.g. $T(\text{`}yptosys\text{'})$ can output $(\text{`}tos\text{'}, (-2, \text{`}ypt\text{'}), (2, \text{`}sys\text{'}))$, but many other tokenizations of the same string are possible. We call k-gram kg_1 an *s-gram* and the remaining k-grams *x-grams*, in parallel to the s-term and x-term terminology of OXT, and as in OXT the s-gram should be chosen as the least frequent k-gram in the tokenization of q. Let KG be a list of k-grams which occur in DB. Let $\mathsf{DB}(\mathsf{kg})$ be the set of $(\mathsf{ind}, \mathsf{pos})$ pairs s.t. $\mathsf{DB}[\mathsf{ind}]$ contains k-gram kg at position pos, and let $\mathsf{DB}(\mathsf{ind}, \mathsf{kg})$ be the set of pos's s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathsf{kg})$.

The basic idea of the above conjuctive-search protocol to handling substrings is that the hashes xtag inserted into the XSet will use PRF F_G applied to a *triple* $(\mathsf{kg}, \mathsf{ind}, \mathsf{pos})$ for each $\mathsf{kg} \in \mathsf{KG}$ and $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathsf{kg})$, and when processing search query q where $T(q) = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$, server $\mathcal{E}$ will return (encrypted) index ind corresponding to some $(\mathsf{ind}_c, \mathsf{pos}_c)$ *pair* in $\mathsf{DB}(\mathsf{kg}_1)$ if and only if

$$F_G(K_G, (\mathsf{kg}_i, \mathsf{ind}_c, \mathsf{pos}_c + \Delta_i)) \in \mathsf{XSet} \text{ for } i = 2, \ldots, \mathsf{h}$$

Setup($\mathsf{DB}, \mathsf{RDK}$)

- Select keys K_S, K_T for PRF F_τ and K_I, K_X for PRF F_p, and parse DB as $(\mathsf{ind}_i, \mathsf{pos}_i, \mathsf{kg}_i)_{i=1}^{d}$. (PRF F_τ maps onto $\{0,1\}^\tau$ and F_p onto $\mathbb{Z}_p$.)
- Initialize $\mathbf{T}$ to an empty array and XSet to an empty set. For each k-gram $\mathsf{kg} \in \mathsf{KG}$ do the following:
  - Set $\mathsf{strap} \leftarrow F_\tau(K_S, \mathsf{kg})$, $(K_z, K_e, K_u) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2), F_\tau(\mathsf{strap}, 3))$.
  - For $c = 1, \ldots, |\mathsf{DB}(\mathsf{kg})|$, for $(\mathsf{ind}, \mathsf{pos})$ a c-th tuple in $\mathsf{DB}(\mathsf{kg})$ (randomly permuted) do:
    - Set $\mathsf{rdk} \leftarrow \mathsf{RDK}(\mathsf{ind})$, $\mathsf{e} \leftarrow \mathsf{Enc}(K_e, (\mathsf{ind}|\mathsf{rdk}))$, $\mathsf{xind} \leftarrow F_p(K_I, \mathsf{ind})$.
    - Set $\mathsf{xtag} \leftarrow g^{F_p(K_X, \mathsf{kg}) \cdot \mathsf{xind}^{\mathsf{pos}}}$ 2and add xtag to XSet.
    - Set $z \leftarrow F_p(K_z, c)$, $u \leftarrow F_p(K_u, c)$, $y \leftarrow \mathsf{xind} \cdot z^{-1}$, $v \leftarrow \mathsf{xind}^{\mathsf{pos}} \cdot u^{-1}$.
    - Append (e, y, v) to $\mathbf{T}[\mathsf{kg}]$.
- Set $\mathsf{TSet} \leftarrow \mathsf{TSetSetup}(\mathbf{T}, \langle F_\tau \rangle, K_T)$. Output $K = (K_S, K_X, K_T)$ and $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$.

Search protocol

Client $\mathcal{C}$, on input $K = (K_S, K_X, K_T)$ defined above and query q s.t. $T(q) = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$:

- Set $\mathsf{stag} \leftarrow F_\tau(K_T, \mathsf{kg}_1)$, $\mathsf{strap} \leftarrow F_\tau(K_S, \mathsf{kg}_1)$.
- $(K_z, K_e, K_u) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2), F_\tau(\mathsf{strap}, 3))$, and $\{\mathsf{xtrap}_i \leftarrow g^{F_p(K_X, \mathsf{kg}_i)}\}_{i=2}^{\mathsf{h}}$.
- Send $(\mathsf{stag}, \Delta_2, \ldots, \Delta_\mathsf{h})$ to $\mathcal{E}$, and for $c = 1, 2, \ldots$, until $\mathcal{E}$ sends stop, do the following:
  - Set $z_c \leftarrow F_p(K_z, c)$, $u_c \leftarrow F_p(K_u, c)$, and $\{\mathsf{xtoken}[c, i] \leftarrow (\mathsf{xtrap}_i)^{((z_c)^{\Delta_i} \cdot (u_c))}\}_{i=2}^{\mathsf{h}}$.
  - Send $\mathsf{xtoken}[c] = (\mathsf{xtoken}[c, 2], \ldots, \mathsf{xtoken}[c, \mathsf{h}])$ to $\mathcal{E}$.

Server $\mathcal{E}$, on input $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet})$, responds with a set ESet formed as follows:

- On message $(\mathsf{stag}, \Delta_2, \ldots, \Delta_n)$ from $\mathcal{C}$, retrieve $\mathbf{t} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$ from TSet.
- For $c = 1, ..., |\mathbf{t}|$, retrieve c-th tuple (e, y, v) in $\mathbf{t}$.
- On $\mathsf{xtoken}[c]$ from $\mathcal{C}$, add e to ESet if $\forall i = 2, \ldots, \mathsf{h} : (\mathsf{xtoken}[c, i])^{(y^{\Delta_i} \cdot v)} \in \mathsf{XSet}$. When $c = |\mathbf{t}|$ send stop to $\mathcal{C}$.

Client $\mathcal{C}$ computes $(\mathsf{ind}|\mathsf{rdk}) \leftarrow \mathsf{Dec}(K_e, \mathsf{e})$ for each e in ESet and adds $(\mathsf{ind}, \mathsf{rdk})$ to its output.

Figure 6.1: SUB-SSE-OXT: SSE Protocol for Substring Search (shadowed text indicates additions to the basic OXT protocol for supporting substring queries)

To support this modified search over encrypted data the setup procedure $\mathsf{Setup}(\mathsf{DB}, \mathsf{RDK})$ forms EDB as a pair of data structures TSet and XSet as in OXT, except that keywords are replaced by k-grams and both the encrypted tuples in TSet and the hashes xtag in XSet will be modified by the position-related information as follows. First, the tuple corresponding to the c-th (index,position) pair $(\mathsf{ind}_c, \mathsf{pos}_c)$ in $\mathsf{DB}(\mathsf{kg})$ will contain value $y_c = F_p(K_I, \mathsf{ind}_c) \cdot F_p(K_z, c)^{-1}$ together with a new position-related value $v_c = F_p(K_I, \mathsf{ind}_c)^{\mathsf{pos}_c} \cdot F_p(K_u, c)^{-1}$, where K_z, K_u are independent PRF keys specific to kg. Secondly, XSet will contain values computed as:

$$F_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}, \mathsf{pos})) = g^{F_p(K_X, \mathsf{kg}) \cdot F_p(K_I, \mathsf{ind})^{\mathsf{pos}}} \tag{6.1}$$

In the Search protocol, client $\mathcal{C}$ will tokenize its query q as $T(q) = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$, send $\mathsf{stag}_{\mathsf{kg}_1} = F_T(K_T, \mathsf{kg}_1)$ to server $\mathcal{E}$, who uses it to retrieve $\mathsf{TSet}(\mathsf{kg}_1)$ from TSet, send the position-shift vectors $(\Delta_2, \ldots, \Delta_\mathsf{h})$ to $\mathcal{E}$, and then, in order for $\mathcal{E}$ to compute $F_G(K_G, (\mathsf{kg}_i, \mathsf{ind}_c, \mathsf{pos}_c + \Delta_i))$ for all c, i pairs, client $\mathcal{C}$ sends to $\mathcal{E}$:

$$\mathsf{xtoken}[c, i] = g^{F_p(K_X, \mathsf{kg}_i) \cdot (F_p(K_z, c))^{\Delta_i} \cdot F_p(K_u, c)}$$

which lets $\mathcal{E}$ compute $F_G(\mathsf{kg}_i, \mathsf{ind}_c, \mathsf{pos}_c + \Delta_i)$ as $(\mathsf{xtoken}[c, i])$ exponentiated to power $(y_c)^{\Delta_i} \cdot v_c$ for (y_c, v_c) in the c-th tuple in $\mathsf{TSet}(\mathsf{kg}_1)$, which computes correctly because

$$y_c^{\Delta_i} \cdot v_c = F_p(K_I, \mathsf{ind}_c)^{\Delta_i + \mathsf{pos}_c} \cdot F_p(K_z, c)^{-\Delta_i} \cdot F_p(K_u, c)^{-1}$$

6.2.2 Wildcards and Phrase Queries

Any sequence of single character wildcards within regular substring queries can be handled by changing tokenization to allow gaps in the query string covered by the computed tokens, e.g. $T('ypt??yst')$ would output $('ypt', (5, 'yst'))$.

In addition to support *wildcard queries* matching prefixes and/or suffixes, we add special "anchor" tokens at the beginning ($'\hat{}'$) and end ($'\$'$) of every record to mark the text boundaries. These anchors are then added during tokenization. This allows searching for substrings at fixed positions within a record. For these queries $T('ypt??yst')$ would output $('\hat{}yp', (1,'ypt'), (6,'yst'), (7,'st\$'))$

Still, this simple change limits us to queries which contain k consecutive characters in-between every substring of wildcards. However, we can remove this restriction if we add to the the XSet all unigrams (i.e. $k = 1$) occurring in a text in addition to the original k-grams.

Adding support for phrase queries is another simple change to the way we parse DB. Instead of parsing by (k-gram,position) pairs, we parse each record by (word,position). Tokenization of q then becomes splitting q into its component words and relative position of each word to the s-term word. As with substrings, wildcards in q result in a gap in the returned Δ's.

6.2.3 Query Flexibility

While many queries can be formed by using substring or wildcard queries independently, many queries are not computable. We can greatly increase the number of available queries by combining the two query types. This allows us to answer any query q s.t. all non-wildcard characters in q are part of at least one k length substring containing no wildcards and q starts and ends with a non-wildcard character. This may require a sufficiently large k (a performance benefit) but limit the type of queries supported. To further increase flexibility we can index fields with multiple values for k or with a different k for each data structure: k_x for XSet and k_s for TSet. The result is a very flexible policy that we can support any query q that meets the following: (1) there exists at least one consecutive k_s length sequence of non-wildcards in q, (2) all non-wildcard characters in q are part of at least one k_x length substring containing no wildcards, and (3) q starts and ends with a non-wildcard character.

Condition (3) above can be avoided if we have an index for $k = 1$ by exploiting OXT's general support of boolean expressions including negation: To handle queries q with n leading (resp. trailing) wildcards, we take the tokenization t of the query string stripped of the leading (resp. trailing) wildcards q' and search for q' but make sure to exclude matches which would be a distance less than n from an anchor. Formally we query: $t \wedge \neg \vee_{i=1}^{n} (-i, \hat{})$ (resp. $t \wedge \neg \vee_{i=1}^{n} (\delta_{\max} + i, \$)$) with $\hat{}$ and $\hat{}$ the anchors and $\delta_{\max}$ the relative position of the right "edge" of q'. The OXT support for general boolean expressions can also be used to support some subset of regular expressions: besides the already implicitly used conjunctions, we could support queries containg: disjunctions to handle alternative sub-patterns such as "Court (Road|Street)", negations to explicitly exclude sub-patterns such as "Michael!(a)" and combinations thereof.

6.2.4 Substring Protocol Extensions

Firstly, we generalize the substring-search protocol SUB-SSE-OXT to support any Boolean query where atomic terms can be formed by any number of substring search terms and/or exact keyword terms.

We note that in above protocol substring/wildcard terms have to be s-terms which restricts the task to handle general queries. Nevertheless, we still can trivially extend above to handle cases where there is at most one substring or wildcard term per conjunction at the "top-level" of the query expression (tree): we just append any additional top-level conjuncts to the substring/wild-card term as x-terms and compose them as in OXT with any other top-level disjuncts.

Furthermore, we note that a pattern matching only a single k-gram (*singleton*) can be treated as a normal equality-match term and for many environments the set of lengths of queried sub-sequences can be fairly small, e.g., in one government sponsored project it was 3. To

exploit this at only moderate cost, we can add a k-gram index for all k values in the set of queryable sub-sequence lengths and include in the index position information as normal and also as information for equality-matching (i.e. we add two tags to the XSet). During query-processing we always try to select a k for a sub-sequence term which results in a singleton and correspondingly compute the equality-term tag rather than the k-gram-tag in such a case. With this strategy, we can handle multiple substring/wildcard terms as long as all but one are singletons per top-level conjuncts. These additional indexes have the added benifit of increased query-time performance.

However, to allow for arbitrary queries we have to extend our protocols. We call the resulting protocol MIXED-SSE-OXT, so named because it freely *mixes* substring and exact keyword search terms, and present it in Section 6.5. The ability to handle Boolean formulas on exact keywords together with substring terms comes from the similarities between substring-handling SUB-SSE-OXT and Boolean-formula-handling OXT of [CJJ$^+$13]. However, one significant adjustment needed to put the two together is to disassociate the position-related information v_c in the tuples in $\mathsf{TSet}(\mathsf{kg})$ from the index-related information y_c in these tuples. This is because when all k-gram terms are x-terms (as would be the the case e.g. when an exact keyword is chosen as an s-term) then $\mathcal{E}$ must identify the position-related information pertaining to a particular $(\mathsf{kg}, \mathsf{ind})$ pair given the $(\mathsf{kg}, \mathsf{ind})$-related xtoken value. Our MIXED-SSE-OXT protocol supports this by adding another oblivious TSet-like datastructure which uses $\mathsf{xtag}_{\mathsf{kg},\mathsf{ind}}$ to retrieve the position-related information, i.e. the v_c's, for all $\mathsf{pos} \in \mathsf{DB}(\mathsf{ind}, \mathsf{kg})$.

A second extension generalizes the SUB-SSE-OXT protocol to the OSPIR setting [JJK$^+$13] where $\mathcal{D}$ can *obliviously* enable third-party clients $\mathcal{C}$ to compute the search-enabling tokens (see Section 6.1). The main ingredient in this extension is the usage of Oblivious PRF (OPRF) evaluation for several PRF functions used in MIXED-SSE-OXT for computing search tokens. Another important component is a novel protocol which securely computes

the $\mathsf{xtag}_{\mathsf{kg},\mathsf{ind},\mathsf{pos}}$ values given these obliviously-generated trapdoors, in a way which avoids leaking any partial-match information to $\mathcal{C}$. This protocol, named MIXED-OSPIR-OXT and presented in Section 6.5.2, uses bilinear maps which results in a significant slowdown compared to the MIXED-SSE-OXT in the (single client) SSE setting.

6.3 Security Analysis

Privacy of an SSE scheme, in the SSE, Multi-Client, or OSPIR settings, is quantified by a *leakage profile* $\mathcal{L}$, which is a function of the database DB and the sequence of client's queries $\mathbf{q}$. We call an SSE scheme $\mathcal{L}$-*semantically-secure* against party P (which can be $\mathcal{C}$, $\mathcal{E}$, or $\mathcal{D}$) if for all DB and $\mathbf{q}$, the entirety of P's view of an execution of the SSE scheme on database DB and $\mathcal{C}$'s sequence of queries $\mathbf{q}$ is efficiently *simulatable* given only $\mathcal{L}(\mathsf{DB}, \mathbf{q})$. We say that the scheme is *adaptively* secure if the queries in $\mathbf{q}$ can be set adaptively by the adversary based on their current view of the protocol execution. An efficient simulation of a party's view in the protocol means that everything that the protocol exposes to this party carries no more information than what is revealed by the $\mathcal{L}(\mathsf{DB}, \mathbf{q})$ function. Therefore specification of the $\mathcal{L}$ function fully characterizes the privacy quality of the solution: What it reveals about data DB and queries $\mathbf{q}$, and thus also what it hides. (See [CJJ+13, JJK+13] for a more formal exposition.)

6.3.1 Security of Substring Queries

Here we prove the security of protocol SUB-SSE-OXT against server $\mathcal{E}$. Our security arguments are based on the following assumptions: the T-set implementation is secure against adaptive adversaries [CJJ+13, CJJ+14]; F_p and F_τ are secure pseudorandom functions; the

hash function H is modeled as a random oracle; and the q-DDH assumption [BB04] (see Section 6.6) holds in the group G.[1]

Security Against Server $\mathcal{E}$. We first describe the leakage function corresponding to server $\mathcal{E}$. It is an adaptation of the leakage for the conjunctive protocol from [CJJ+13] to our setting. To simplify presentation (avoiding complex notation) and focus on the important aspects of this leakage function, our description assumes that substring queries contain a single substring tokenized into two k-grams, i.e., one s-term k-gram and one x-term k-gram. The extension to the general case is similar to the extension from two-term conjunctions to general conjunctions in [CJJ+13].

Leakage to Server $\mathcal{E}$. We represent a sequence of Q non-adaptive substring queries by $\mathbf{q} = (\mathbf{s}, \mathbf{x}, \mathbf{\Delta})$ s.t. $(\mathbf{s}[i], (\mathbf{x}[i], \mathbf{\Delta}[i]))$ is the tokenization $T(\mathbf{q}[i])$ of the i-th substring query $\mathbf{q}[i]$, where $\mathbf{s}[i], \mathbf{x}[i]$ are k-grams, and $\mathbf{\Delta}[i]$ is an integer between $-k+1$ and $k-1$. For notation simplicity we assume that vector $\mathbf{q}$ does not contain repeated queries, although $\mathcal{E}$ would learn that a repeated query has been made. Function $\mathcal{L}_{\mathcal{E}}(\mathsf{DB}, \mathbf{q})$ which specifies leakage to $\mathcal{E}$ outputs $(N, \bar{\mathbf{s}}, \mathsf{SP}, \mathsf{RP}, \mathsf{DP}, \mathsf{IP})$ defined as follows:

• The $(N, \bar{\mathbf{s}}, \mathsf{SP}, \mathsf{RP})$ part of this leakage is exactly the same as in the conjunctive SSE protocol SSE-OXT of [CJJ+13] on which our substring-search SUB-SSE-OXT protocol is based. $N = \sum_{i=1}^{d} |\mathsf{W}_i|$ is the total number of appearances of all k-grams in all the documents, and it is revealed simply by the size of the EDB metadata. $\bar{\mathbf{s}} \in [m]^Q$ is the *equality pattern* of $\mathbf{s} \in \mathsf{KG}^Q$ indicating which queries have the equal s-terms. For example, if $\mathbf{s} = (abc, abc, xyz, pqr, abc, pqr, def, xyz, pqr)$ then $\bar{\mathbf{s}} = (1, 1, 2, 3, 1, 3, 4, 2, 3)$. SP is the *s-term support size* which is the number of occurrences of the s-term k-gram in the database, i.e. $\mathsf{SP}[i] = |\mathsf{DB}(\mathbf{s}[i])|$. Finally, RP is the *results pattern*, i.e. $\mathsf{RP}[i]$ is the set of $(\mathsf{ind}, \mathsf{pos})$ pairs

[1] The extension to the OSPIR model also assumes the One-More Gap Diffie-Hellman assumption and assumes bilinear groups where the linear DH assumption [BBS04, Sha07] holds.

where ind is an identifier of document which matches the query q, and pos is a position of the s-term k-gram $\mathbf{s}[i]$ in that document.

• DP is the *Delta pattern* $\boldsymbol{\Delta}[i]$ of the queries, i.e. the shifts between k-grams in a query which result from the tokenization of the queries.

• IP is the *conditional intersection pattern*, which is a Q by Q table IP defined as follows: $\mathsf{IP}[i,j] = \emptyset$ if $i = j$ or $\mathbf{x}[i] \neq \mathbf{x}[j]$. Otherwise, $\mathsf{IP}[i,j]$ is the set of all triples $(\mathsf{ind}, \mathsf{pos}, \mathsf{pos}')$ (possibly empty) s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathbf{s}[i])$, $(\mathsf{ind}, \mathsf{pos}') \in \mathsf{DB}(\mathbf{s}[j])$, and $\mathsf{pos}' = \mathsf{pos} + (\boldsymbol{\Delta}[i] - \boldsymbol{\Delta}[j])$.

Understanding Leakage Components. Parameter N is the size of the meta-data, and leaking such a bound is unavoidable. The equality pattern $\overline{\mathbf{s}}$, which leaks repetitions in the s-term k-gram of different substring queries, and the s-term support size SP, which leaks the total number of occurrences of this s-term in the database, are both a consequence of the optimized search that singles out the s-term in the query, which we adopt from the conjunctive SSE search solution of [CJJ+13]. RP is the result of the query and therefore no real leakage in the context of SSE. Note also that the RP over-estimates the information $\mathcal{E}$ observes, because $\mathcal{E}$ observes only a pointer to the encrypted document, and a pointer to the encrypted tuple storing a unique $(\mathsf{ind}, \mathsf{pos})$ pair, but not the pair $(\mathsf{ind}, \mathsf{pos})$ itself. DP reflects the fact that our protocols leak the relative shifts Δ between k-grams which result from tokenization of the searched string. If tokenization was canonical, and divided a substring into k-grams based only on the substring length, the shifts Δ would reveal only the substring length. (Otherwise, see below for how Δ's can be hidden from $\mathcal{E}$.)

The IP component is the most subtle. It is a consequence of the fact that when processing the $\mathbf{q}[i]$ query $\mathcal{E}$ computes the (pseudo)random function $F_G(\mathbf{x}[i], \mathsf{ind}, \mathsf{pos}+\boldsymbol{\Delta}[i])$ for all $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathbf{s}[i])$, and hence can see collisions in it. Consequently, if two queries $\mathbf{q}[i]$ and $\mathbf{q}[j]$ have the same x-gram then for any document ind which contains the s-grams $\mathbf{s}[i]$ and $\mathbf{s}[j]$ in

positions, respectively, pos and $\mathsf{pos}' = \mathsf{pos} + (\mathbf{\Delta}[i] - \mathbf{\Delta}[j])$, server $\mathcal{E}$ can observe a collision in F_G and triple $(\mathsf{ind}, \mathsf{pos}, \mathsf{pos}')$ will be included in the IP leakage. Note, however, that $\mathsf{IP}[i, j]$ defined above overstates this leakage, because $\mathcal{E}$ does not learn the $\mathsf{ind}, \mathsf{pos}, \mathsf{pos}'$ values themselves, but only establishes a link between two *encrypted* tuples, one containing $(\mathsf{ind}, \mathsf{pos})$ in $\mathsf{TSet}(\mathbf{s}[i])$ and one containing $(\mathsf{ind}, \mathsf{pos}')$ in $\mathsf{TSet}(\mathbf{s}[j])$. To visualize the type of queries which will trigger this leakage, take $k = 3$, $\mathbf{q}[i]$ = *MOTHER*, $\mathbf{q}[j]$ = *OTHER*, and let $\mathbf{q}[i]$ and $\mathbf{q}[j]$ tokenize with a common x-gram, e.g. $T(\mathbf{q}[i]) = (\text{MOT}, (\text{HER}, 3))$ and $T(\mathbf{q}[j]) = (\text{OTH}, (\text{HER}, 2))$. The $\mathsf{IP}[i, j]$ leakage will contain tuple $(\mathsf{ind}, \mathsf{pos}, \mathsf{pos}')$ for $\mathsf{pos}' = \mathsf{pos} + (\mathbf{\Delta}[i] - \mathbf{\Delta}[j]) = \mathsf{pos} + 1$ iff record $\mathsf{DB}[\mathsf{ind}]$ contains 3-gram $\mathbf{s}[i] = \text{MOT}$ at position pos and 3-gram $\mathbf{s}[j] = \text{OTH}$ at position $\mathsf{pos} + 1$, i.e. iff it contains substring MOTH. The IP leakage can be seen as the price we pay for the rich functionality enabled by our x-terms and XSet approach that allows for the computation of arbitrary Boolean queries. This leakage follows from the fact that given a query q s.t. $T(q) = (s, (x, \Delta))$, our protocol allows $\mathcal{E}$ to learn the value of function $F_G(x, \mathsf{ind}, \mathsf{pos}+\Delta)$ for every $(\mathsf{ind}, \mathsf{pos})$ in $\mathsf{DB}(s)$. Therefore if two queries $\mathbf{q}[i] = (\mathbf{s}[i], \mathbf{x}[i], \mathbf{\Delta}[i])$ and $\mathbf{q}[j] = (\mathbf{s}[j], \mathbf{x}[j], \mathbf{\Delta}[j])$ have the same x-terms $\mathbf{x}[i] = \mathbf{x}[j] = x$, $\mathcal{E}$ could compare the results of this function for every $(\mathsf{ind}_i, \mathsf{pos}_i)$ in the Tset $\mathsf{DB}(\mathbf{s}[i])$ and every $(\mathsf{ind}_j, \mathsf{pos}_j)$ in $\mathsf{DB}(\mathbf{s}[j])$ and discover if $F(x, \mathsf{ind}_i, \mathsf{pos}_i + \mathbf{\Delta}[i]) = F(x, \mathsf{ind}_j, \mathsf{pos}_j + \mathbf{\Delta}[j])$, which by the collision-resistant property of F_G implies that $\mathsf{ind}_i = \mathsf{ind}_j$.

Theorem 6.1. *Protocol SUB-SSE-OXT (restricted to substrings which tokenize into two k-grams) is adaptively $\mathcal{L}_{\mathcal{E}}$-semantically-secure against malicious server $\mathcal{E}$, assuming the security of the PRF's, the encryption scheme* Enc*, and the* TSet *scheme, the random oracle model for hash functions, and the q-DDH assumption on the group G of prime order.*

The proof of Theorem 6.1 is included in Section 6.6.

Hiding Deltas. Since the tokenizer T should pick the least frequent k-gram as an s-gram, the information on which k-gram was chosen, which is visible from the vector of Δ's, can leak some sensitive statistics about the substring term. For example, if the tokenizer chooses the

s-gram based on the k-gram frequency statistics, but then determines all the x-grams in a canonical way, then there are $n-k+1$ ways of tokenizing an n-character substring, hence $\mathcal{E}$ learns to which of the $n-k+1$ partitions the client's substring term belongs. If this moderate information leakage is unacceptable, it can be eliminated entirely at a moderate cost incurred by a Δ-hiding variant of the Search protocol. This can be done by relying on a multiplicative homomorphism of either ElGamal or linear encryption to create a multiplicative sharing of xind^{Δ} without revealing Δ to $\mathcal{E}$, and then combine it with the multiplicative sharing of $\mathsf{xind}^{\mathsf{pos}}$ in the xtag computation.

6.4 Implementation and Performance

Here we provide testing and performance information for our prototype implementation of the SUB-SSE-OXT protocol described in Section 6.2.1. The results confirm the scalability of our solutions to very large databases and complex queries. The prototype is an extension of the OXT implementation of [CJJ+14]. Both the description of the changes and performance information are limited, to the extent possible, to the protocols introduced in this chapter.

Prototype Summary. The three components of our system are the preprocessor, the server, and the client. The preprocessor generates the encrypted database from the cleartext data. The client, which implements a representative set of SQL commands, 'encrypts' end-user requests and 'decrypts' server responses. The server uses the encrypted database to answer client SELECT-type queries or expands the encrypted database on UPDATE, INSERT, and (even) DELETE queries [CJJ+14].

Support for substring and wildcard queries required redesigning pre-processing to take into account the k-gram position information, adding support for 'k-gram'-based record tokenization to the client, and changing the Search protocol to support *position-enhanced* computation

(see Section 6.2) and authorization. A few other changes were necessary in order to continue handling UPDATE, INSERT and DELETE queries. These extensions largely follow the update mechanics outlined in [CJJ+14], with the addition of a new PSet^+ data structure.

To match the SQL standard, our implementation uses the `LIKE` operator syntax for substring and wildcard queries: `'_'` (`'%'`) represent single-character (variable-length) wildcards and the query must match the complete field, i.e, unless a query must match the prefix (suffix) of fields, it should begin (end) with a `'%'`.

Experimental Platform. The experiments described in the remainder of this section were run on two Dell PowerEdge R710 systems, each one of them equipped with two Intel Xeon X5650 processors, 96GB RAM (12x8 1066MHz), an embedded Broadcom 1GB Ethernet with TOE and a PERC H700 RAID controller with a 1GB Non-Volatile Cache and 1 or 2 daisy-chained MD1200 disk controllers each with 12 2TB 7.2k RPM Near-Line SAS hard drives configured for Raid 6 (19TB and 38TB total storage per machine).

An automated test harness, written by an independent evaluator [VPH+15], drives the evaluation, including the set of queries and the dataset used in the experiments.

Dataset. The synthetic dataset used in the reported experiments is a US census-like table with twenty one columns of standard personal information, such as name (first, last), address (street, city, state, zipcode), SSN, etc. The values in each column are generated according to the distributions in the most recent US census. In addition, the table has one XML column with at most 10000 characters, four text columns with varying average lengths (a total of at most 12300 characters or $\approx$ 2000 words), and a binary column (payload) with a maximum size of 100KB. Our system can perform structured queries on data in all but the XML and binary columns. The size of (number of records in) the table is a parameter of the dataset generator. We tested on a wide variety of database sizes, but we focus our results on a table with 100 million records or 10TBytes.

Experimental Methodology. In the initial step, the encrypted database is created from the cleartext data stored in a MariaDB (a variant of open-source MySQL RDBMS) table. Then, a per-protocol collection of SQL queries, generated by the harness to test its features, is run against the MariaDB sever and against our system. The queries are issued sequentially by the harness, which also records the results and the execution times of each query. Finally, the harness validates the test results by comparing the result sets from our system and from the MariaDB server. Not only does this step validate the correctness of our system, it also ensures our system meets our theoretical false positive threshold over large, automatically generated, collections of queries.

Encrypted Index. We built a searchable index on all personal information columns (twenty one) in the plaintext database but we only use a small subset of these indexes for the following experiments. Note that we support substring and wildcard queries simultaneously over a given column using a single shared index. We built a substring-wildcard index for four columns (average length of 12 characters) and a range index for five columns of varying types (one 64 bit integer, one date, one 32 bit integer, and one enum). Each substring-wildcard index was constructed with a single k value of 4. Each range index has a granularity of one. For the date type, this equates to a day. We support date queries between 0-01-01 and 9999-12-31, and integer queries between 0 and integer max ($2^{32} - 1$ or $2^{64} - 1$).

On average each record generates 256.6 document-keyword pairs (tuples) among all indexes. This equates to a total encrypted index for our largest database of $\approx$ 20TB. We back our XSet by an in memory Bloom filter with a false positive rate of 2^{-12}; this allows us to save unnecessary disk accesses and it does not influence the false positive rate of the system.

Performance Costs by Query Type. Our complex query types have both increased storage overhead and query time costs as compared to the keyword only implementation of [CJJ+14]. In order to support substring and wildcard queries on a column, we must store

additional tuples: for a record of length l (for the indexed field) we must store $(l - k) + 3$ tuples. Note that we must pay this cost for each k we chose to create the index for. The choice of k also affects query time performance. For a query q, it's performance is linearly dependent on the number of tokens generated by the tokenization $T(q)$. A smaller k results in a larger number of tokens. Specifically for subsequence queries there will be $\lceil |q|/k \rceil$-1 xtokens[2]. k also impacts the number of matching documents returned by the s-term. A larger k results in a higher entropy s-term. The choice of k is a careful trade-off between efficiency and flexibility.

Phrase queries incur storage costs linear in the total number of words in a column. Specifically for every record with n free-text words, the index stores n tuples. Although phrase queries and free-text queries can be supported via the same index, we have to pay the marginally higher price of the phrase index in which we must store even repeated words.

Encrypted Search Performance. We illustrate the performance of our system using the latency (i.e., total time from query issuing to completion) of a large number of representative `SELECT` queries. The independent evaluator selected a representative set of queries to test the correctness and performance of the substring and wildcard queries (phrase queries were not implemented). The two leftmost columns in Table 6.1 show how many unique queries were selected for each query type. The third, fourth and fifth columns characterize the 95% fastest queries of each type. Finally, the rightmost column shows the percentage of queries that complete in less than two minutes.

All queries follow the pattern `SELECT id FROM CensusTable WHERE ...`, with each query having a specific WHERE clause. Specific queries were chosen to assess the performance effect of differing result set sizes.

[2]Wildcard queries pay a similar overhead, related to the size of each contiguous substring within the query.

Query type	# of queries	fastest 95% avg	min	max	% ≤ 120 secs
range	197	.37	.19	.61	100
substring	939	40	0.22	166	93
wildcard	511	31.22	6.7	224	93

Table 6.1: Latency (in secs) for 10 TByte DB, 100M records, 25.6 billion record-keyword pairs

Our instantiation of SUB-SSE-OXT includes extensions for supporting substring and wildcard searches simultaneously. However, to evaluate the effects of each specific extension we measure them individually. Both query types use the `LIKE` operator in the `WHERE` clause.

Substring queries use the variable-length wildcard '%' at the beginning, at the end, or at both ends of the `LIKE` operand, as in `WHERE city LIKE '%ttle Falls%'`. Wildcard queries use the single-character wildcard ('_') anywhere in the `LIKE` operand, provided the query criteria dictated by k is still met.

In addition, we noticed that the choice of s-gram dominates the latency of the substring queries. Our analysis shows that low performing queries can often be tied to high-frequency s-terms (e.g., "ing " or "gton "), which are associated with large Tsets. By default, the current implementation uses the first k characters in the pattern string as s-gram. Thus, implementing a tokenization strategy guided by the text statistics (which we leave for future work) can significantly reduce query latency for many of the slow performers. To estimate the potential benefits of such a strategy, we added the `STARTAT 'n'` option to the `LIKE 'pattern'` operator, where `'n'` is the starting position of the s-gram. Experiments using the `'%gton Colle%'` pattern show latency improvements of up to 32 times when the s-gram starts at the third or fourth character in the pattern string.

Comparison to Cleartext Search. Here we include the most relevant aspects of the performance comparison between our prototype and MariaDB. In the case of the 100 mil-

lion record database, for $\approx 45\%$ of the range queries, the two systems have very similar performance. For the remaining 55%, our system is increasingly (up to 500 times!) faster. The large variations in MariaDB performance seem to arise from its reliance on data (and index) caching, which is hindered by large DBs. In contrast, our system issues between $\log_2 s$ and $2\log_2 s$ disk accesses *in parallel* (where s is the size of the cover). On smaller census databases (with fewer records) that fit in RAM, MariaDB outperforms our system, sometimes by more than one order of magnitude, although in this case all query latencies (ours and MariaDB's) are under a second. Additionally, for substring and wildcard queries and the largest, 100 million records, database our system always outperforms MariaDB, admittedly due to MariaDB's lack of support for a specialized-index based substring search. Instead, it often scans the dataset to resolve queries involving the `LIKE` operator.

6.5 Substring SSE Extensions

6.5.1 MIXED-SSE-OXT: Substring Terms in General Boolean Formula Queries

We show how to generalize the SUB-SSE-OXT protocol so that it supports conjunctions (or indeed, any Boolean formula) whose atomic terms can be formed by any number of substring search terms and/or exact keyword terms, with flexible choice of s-term as either one of the exact keyword terms or a k-gram in one of the substring terms. The resulting protocol, shown in Figure 6.2, is called MIXED-SSE-OXT because it freely *mixes* substring and exact keyword search terms.

Protocol SUB-SSE-OXT stores in each tuple of $\mathbf{T}[\mathsf{kg}]$ the (encrypted) values of xind and $\mathsf{xind}^{\mathsf{pos}}$. In MIXED-SSE-OXT we decouple these two values: We store in $\mathbf{T}[\mathsf{kg}]$ only the xind values, and we create a separate data structure for the (encrypted) position-related

Setup($\mathsf{DB}, \mathsf{RDK}$)

- Pick keys K_S, K_T for PRF F_τ, K_I, K_X for PRF F_p, parse DB as $(\mathsf{ind}_i, \mathsf{W}_i)_{i=1}^{d}$ and $(\mathsf{ind}_i, \mathsf{pos}_i, \mathsf{kg}_i)_{i=1}^{d'}$.
- Initialize $\mathbf{T}$ and $\mathbf{P}$ to empty arrays and XSet to an empty set. For each $w \in \mathsf{W} \cup \mathsf{KG}$ do the following:
  - Set $\mathsf{strap} \leftarrow F_\tau(K_S, w)$ and $(K_z, K_e) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$.
  - For $c = 1, \ldots, |\mathsf{DB}(w)|$, for ind a c-th element in $\mathsf{DB}(w)$ (randomly permuted) do:
    - Set $\mathsf{rdk} \leftarrow \mathsf{RDK}(\mathsf{ind})$, $\mathsf{e} \leftarrow \mathsf{Enc}(K_e, (\mathsf{ind}|\mathsf{rdk}))$, $\mathsf{xind} \leftarrow F_p(K_I, \mathsf{ind})$.
    - Set $z \leftarrow F_p(K_z, c)$, $y \leftarrow \mathsf{xind} \cdot z^{-1}$, and append (e, y) to $\mathbf{T}[w]$.
    - If $w \in \mathsf{W}$ then set $\mathsf{xtag} \leftarrow g^{F_p(K_X, w) \cdot \mathsf{xind}}$ and add xtag to XSet.
    - If $w \in \mathsf{KG}$ then for $c' = 1, \ldots, |\mathsf{DB}(\mathsf{ind}, w)|$, for pos a c'-th element in $\mathsf{DB}(\mathsf{ind}, w)$ do:
      - Set $\mathsf{ptag} \leftarrow g^{F_p(K_X, w) \cdot \mathsf{xind}}$, $K_u \leftarrow F_\tau(\mathsf{strap}, 3, \mathsf{ptag})$, $u \leftarrow F_p(K_u, c')$, $v \leftarrow \mathsf{xind}^{\mathsf{pos}} \cdot u^{-1}$, append v to $\mathbf{P}[(w, \mathsf{ind})]$.
      - Set $\mathsf{xtag} \leftarrow g^{F_p(K_X, w) \cdot F_p(K_I, \mathsf{ind})^{\mathsf{pos}}}$ and add xtag to XSet.
- Set $\mathsf{TSet} \leftarrow \mathsf{TSetSetup}(\mathbf{T}, \langle F_\tau \rangle, K_T)$ and $\mathsf{PSet} \leftarrow \mathsf{TSetSetup}(\mathbf{P}, (K_X, K_I))$.
- Output $K = (K_S, K_X, K_T)$, $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, \mathsf{PSet})$.

Search protocol

- Client $\mathcal{C}$, on input key $K = (K_S, K_X, K_T)$ and conjunctive query $\bar{w}$ consisting of a single (for simplicity) substring-search term q, tokenized as $(\mathsf{kg}_1, (\mathsf{kg}_2, \Delta_2), \ldots, (\mathsf{kg}_\mathsf{h}, \Delta_\mathsf{h}))$, and exact-keyword terms $w_1, \ldots, w_n$, with w_1 as the s-term of the query:
  - Set $\mathsf{stag} \leftarrow F_\tau(K_T, w_1)$, $\mathsf{strap} \leftarrow F_\tau(K_S, w_1)$, $(K_z, K_e) \leftarrow (F_\tau(\mathsf{strap}, 1), F_\tau(\mathsf{strap}, 2))$.
  - Set $\{\mathsf{xtrap}_i \leftarrow g^{F_p(K_X, w_i)}\}_{i=2}^{n}$, $\{\mathsf{xtrap}_{\mathsf{kg}_i} \leftarrow g^{F_p(K_X, \mathsf{kg}_i)}\}_{i=1}^{\mathsf{h}}$, and $\mathsf{strap}_{\mathsf{kg}_1} \leftarrow F_\tau(K_S, \mathsf{kg}_1)$.
  - Send $(\mathsf{stag}, \Delta_2, \ldots, \Delta_\mathsf{h})$ to $\mathcal{E}$.
  - For $c = 1, 2, \ldots$, until $\mathcal{E}$ sends stop_c do:
    - Set $z_c \leftarrow F_p(K_z, c)$; Set $\{\mathsf{xtoken}[c, i] \leftarrow (\mathsf{xtrap}_i)^{z_c}\}_{i=2}^{n}$ and $\mathsf{ptoken}[c] \leftarrow (\mathsf{xtrap}_{\mathsf{kg}_1})^{z_c}$.
    - Send $\mathsf{xtoken}[c] = (\mathsf{xtoken}[c, 2], \ldots, \mathsf{xtoken}[c, n], \mathsf{ptoken}[c])$ to $\mathcal{E}$.
    - On $\mathsf{ptag}[c]$ from $\mathcal{E}$, set $K_u \leftarrow F_\tau(\mathsf{strap}_{\mathsf{kg}_1}, 3, \mathsf{ptag}[c])$, and for $c' = 1, 2, \ldots$, until $\mathcal{E}$ sends $\mathsf{stop}_{c,c'}$ do:
      - Set $u_{c'} \leftarrow F_p(K_u, c')$ and $\left\{\mathsf{xtoken}_{\mathsf{kg}}[c, c', i] \leftarrow (\mathsf{xtrap}_{\mathsf{kg}_i})^{(z_c)^{\Delta_i} \cdot (u_{c'})}\right\}_{i=2}^{\mathsf{h}}$.
      - Send $\mathsf{xtoken}_{\mathsf{kg}}[c, c'] = (\mathsf{xtoken}_{\mathsf{kg}}[c, c', 2], \ldots \mathsf{xtoken}_{\mathsf{kg}}[c, c', \mathsf{h}])$ to $\mathcal{E}$.
- Server $\mathcal{E}$, on input $\mathsf{EDB} = (\mathsf{TSet}, \mathsf{XSet}, \mathsf{PSet})$, responds with a set ESet formed as follows:
  - On $(\mathsf{stag}, \Delta_2, \ldots, \Delta_\mathsf{h})$ from $\mathcal{C}$, retrieve $\mathbf{t} \leftarrow \mathsf{TSetRetrieve}(\mathsf{TSet}, \mathsf{stag})$.
  - For $c = 1, ..., |\mathbf{t}|$, on $\mathsf{xtoken}[c]$ from $\mathcal{C}$, retrieve c-th tuple (e, y) in $\mathbf{t}$, set $(\mathsf{OK}_c^1, \mathsf{OK}_c^2) \leftarrow (0, 0)$.
    - Set $\mathsf{OK}_c^1 \leftarrow 1$ if $\forall i = 2, \ldots, n : (\mathsf{xtoken}[c, i])^y \in \mathsf{XSet}$.
    - Set $\mathsf{ptag}[c] \leftarrow \mathsf{ptoken}[c]^y$, retrieve $\mathbf{p} \leftarrow \mathsf{PSet}[\mathsf{ptag}[c]]$, send $\mathsf{ptag}[c]$ to $\mathcal{C}$.
    - If $|\mathbf{p}| = 0$ send $\mathsf{stop}_{c,c'}$ to $\mathcal{C}$ and continue (to next c). Otherwise, for $c' = 1, \ldots, |\mathbf{p}|$ do:
      - On $\mathsf{xtoken}[c, c']$ from $\mathcal{C}$, retrieve c'-th element v from $\mathbf{p}$;
      - Set $\mathsf{OK}_c^2 \leftarrow 1$ if $\forall i = 2, \ldots, \mathsf{h} : (\mathsf{xtoken}_{\mathsf{kg}}[c, c', i])^{y^{\Delta_i} \cdot v} \in \mathsf{XSet}$; Send $\mathsf{stop}_{c,c'}$ to $\mathcal{C}$ when $c' = |\mathbf{p}|$.
    - If $(\mathsf{OK}_c^1, \mathsf{OK}_c^2) = (1, 1)$ then add e to ESet. When $c = |\mathbf{t}|$ send stop_c to $\mathcal{C}$.
- Client $\mathcal{C}$ computes $(\mathsf{ind}|\mathsf{rdk}) \leftarrow \mathsf{Dec}(K_e, \mathsf{e})$ for each e in ESet, and adds $(\mathsf{ind}, \mathsf{rdk})$ to its output.

Figure 6.2: MIXED-SSE-OXT: SSE for Conjunctions of Multiple Substring and Exact Keyword Terms

values $\mathsf{xind}^{\mathsf{pos}}$. By shifting the encrypted $\mathsf{xind}^{\mathsf{pos}}$ values to another data structure, we can combine the k-gram and keyword indexes together so that for all $a \in (\mathsf{KG} \cup \mathsf{W})$ and all $\mathsf{ind} \in \mathsf{DB}(a)$, list $\mathbf{T}[a]$ will include an entry (y, e) at some position c s.t. $\mathsf{e} = \mathsf{Enc}(K_e, (\mathsf{ind}|\mathsf{rdk}))$ and $y = \mathsf{xind}/z$, for $\mathsf{xind} = F_p(K_I, \mathsf{ind})$, $\mathsf{rdk} = \mathsf{RDK}[\mathsf{ind}]$, $z = F_p(K_z, c)$, and keys K_z, K_e derived from $\mathsf{strap} = F_\tau(K_S, a)$ similarly as in the SUB-SSE-OXT. Treating the k-gram and exact keywords in this uniform way allows us to compose the substring search capability of SUB-SSE-OXT with the Boolean search on exact keywords of SSE-OXT.

For storing the position-related information for resolving subsequence queries, namely the encrypted $\mathsf{xind}^{\mathsf{pos}}$ values, we use a separate look-up table $\mathbf{P}$. Let F'_G denote a "truncated" version of function F_G from equation (6.1), namely

$$F'_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind})) = g^{F_p(K_X, \mathsf{kg}) \cdot F_p(K_I, \mathsf{ind})} \tag{6.2}$$

For every $(\mathsf{kg}, \mathsf{ind})$ s.t. k-gram kg appears in $\mathsf{DB}[\mathsf{ind}]$, $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$ stores a list of values $v = \mathsf{xind}^{\mathsf{pos}}/u$, where $\mathsf{xind} = F_p(K_I, \mathsf{ind})$, for each pos s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathsf{kg})$. The value u that masks c-th value $\mathsf{xind}^{\mathsf{pos}}$ is computed as $u = F_p(K_u, c)$ where $K_u = F_\tau(\mathsf{strap}_{\mathsf{kg}}, \mathsf{ptag}_{(\mathsf{kg},\mathsf{ind})})$ and $\mathsf{ptag}_{(\mathsf{kg},\mathsf{ind})} = F'_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}))$ (ptag serves a similar purpose as stag but for positioning information). Finally, we store $\mathbf{P}$ in another instance of the TSet data structure, called PSet, where a handle for identification and decryption of a list $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$ is an output of $F'_G((K_X, K_I), \cdot)$ on $(\mathsf{kg}, \mathsf{ind})$, i.e. $\mathsf{ptag}_{(\mathsf{kg},ind)}$.

Lastly, the data-structure XSet will store the xtag values for both exact keywords and for (k-gram,position) pairs, i.e. it will contain values $F_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}, \mathsf{pos}))$ for all $\mathsf{kg} \in \mathsf{KG}$ and $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(\mathsf{kg})$ and values $F'_G((K_X, K_I)(w, \mathsf{ind})$ for all $w \in \mathsf{W}$ and $\mathsf{ind} \in \mathsf{DB}(w)$.

The three data structures $(\mathsf{TSet}, \mathsf{PSet}, \mathsf{XSet})$ are used in Search to combine the substring processing in SUB-SSE-OXT with the Boolean search (on exact keywords) of the original SSE-OXT. Assume that $\mathcal{C}$'s query q is a conjunction of n exact query terms $w_1, \ldots, w_n$ and

a single substring search term q' tokenized as $T(q') = (\mathsf{kg}_1, (\Delta_2, \mathsf{kg}_2), \ldots, (\Delta_\mathsf{h}, \mathsf{kg}_\mathsf{h}))$. Assume also that the exact keyword w_1 is chosen as an s-term. All these assumptions are not necessary and are used solely to simplify the protocol description below. Client $\mathcal{C}$ sends $\mathsf{stag}_{w_1} = F_T(K_T, w_1)$ to $\mathcal{E}$, who uses it to retrieve $\mathbf{t} = \mathbf{T}[w_1]$ from TSet. For each (encrypted) ind in $\mathbf{t}$, $\mathcal{C}$ and $\mathcal{E}$ perform the following: First they compute (in parallel, and following the combined operations of SSE-OXT and SUB-SSE-OXT) values $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})} = F'_G((K_X, K_I), (\mathsf{kg}_1, \mathsf{ind}))$ and $\mathsf{xtag}_{(w_i,\mathsf{ind})} = F'_G((K_X, K_I), (w_i, \mathsf{ind}))$ for $i = 2, \ldots, n$. If $\mathsf{xtag}_{(w_i,\mathsf{ind})} \notin \mathsf{XSet}$ for any $i = 2, \ldots, n$ or if the list $\mathbf{p} = \mathbf{P}[\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}]$ is empty, we can conclude that $\mathsf{ind} \notin \mathsf{DB}(q)$, and so $\mathcal{E}$ moves on to the next (encrypted) ind in $\mathbf{t}$. Otherwise, $\mathcal{E}$ sends back $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}$ to $\mathcal{C}$, who uses it to derive the key K_u, and then for each (encrypted) $\mathsf{ind}^\mathsf{pos}$ value in $\mathbf{p}$, $\mathcal{C}$ and $\mathcal{E}$ jointly compute $\mathsf{xtag}_{(\mathsf{kg}_i,\mathsf{ind},\mathsf{pos}+\Delta_i)} = F_G((K_X, K_I), (\mathsf{kg}_i, \mathsf{ind}, \mathsf{pos} + \Delta_i))$ for $i = 2, \ldots, \mathsf{h}$. The latter computation is similar to the one described for SUB-SSE-OXT above, except that the (encrypted) ind value comes from list $\mathbf{t}$ while (encrypted) $\mathsf{ind}^\mathsf{pos}$ value comes from list $\mathbf{p}$. If $\mathsf{xtag}_{(\mathsf{kg}_i,\mathsf{ind},\mathsf{pos}+\Delta_i)} \in \mathsf{XSet}$ for *some* pos in the $\mathbf{p}$ list and all $i = 2, \ldots, \mathsf{h}$, we can conclude (except for probability of collision in F_G) that substring q' appears in $\mathsf{DB}[\mathsf{ind}]$ at position pos, and hence that $\mathsf{ind} \in \mathsf{DB}(q)$. Therefore in that case $\mathcal{E}$ sends ciphertext e corresponding to this ind to $\mathcal{C}$, which allows $\mathcal{C}$ to retrieve and decrypt record $\mathsf{DB}[\mathsf{ind}]$.

If query q involves more substring terms, each of them is processed as the substring term q' above. Since $\mathbf{T}[w]$ for $w \in \mathsf{W}$ and $\mathbf{T}[\mathsf{kg}]$ for $\mathsf{kg} \in \mathsf{KG}$ are implemented in the same way, an s-gram kg_1 from any substring search term can play the role of the s-term. Finally, the protocol can be easily modified to support any query expressed as $w \wedge \Phi(w_1, \ldots, w_n, q'_1, \ldots, q'_k)$, where w is either an exact keyword term or a substring term, $w_1, \ldots, w_n$ are exact keyword terms, $q'_1, \ldots, q'_k$ are substring terms, and Φ is any Boolean formula. The protocol cost is upper-bounded by $(n + \mathsf{h}_1 + \ldots + \mathsf{h}_k)$ exponentiations per party per each tuple in $\mathbf{T}[w]$, where h_i is the number of k-grams in the tokenization of q'_i.

Note that MIXED-SSE-OXT adds an extra communication round compared to SUB-SSE-OXT.However, $\mathcal{E}$ can generate its responses $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}$ (one for each $c = 1, \ldots, |\mathbf{t}|$ and each substring term q_i, $i = 1, \ldots, k$) without retrieving list $\mathsf{PSet}[(\mathsf{kg}_1, \mathsf{ind})]$ from the disk (except for a small probability of a false positive error) if $\mathcal{E}$ keeps a Bloom filter which is small enough to fit in the memory and which allows $\mathcal{E}$ to check if any ptag value corresponds to a non-empty list in PSet.

6.5.2 MIXED-OSPIR-OXT: Substring and Keyword Search in OSPIR Setting

The MIXED-SSE-OXT protocol extends to the Multi-Client and OSPIR settings. Because of the similarity between MIXED-SSE-OXT with the SSE-OXT protocol of [CJJ+13], we can re-use all the techniques of [JJK+13], which adopted protocol SSE-OXT to the Multi-Client and OSPIR settings. Here we recall these techniques briefly: First, we modify several PRF's used by $\mathcal{C}$ in MIXED-SSE-OXT so that they can be efficiently computed via an Oblivious PRF (OPRF) protocol between $\mathcal{C}$ and the data owner $\mathcal{D}$. (In particular we replace g^{xtrap} for $\mathsf{xtrap} = F_p(K_I, \mathsf{kg})$ with $\mathsf{xtrap} = H(\mathsf{kg})^{K_I}$, so e.g. $F_G(\mathsf{kg}, \mathsf{ind}, \mathsf{pos})$ becomes $\mathsf{xtrap}_{\mathsf{kg}}^{F_p(K_I,ind)^{\mathsf{pos}}}$.) The security of the OPRF protocol implies that all the individual terms in $\mathcal{C}$'s query are hidden from $\mathcal{D}$. However, just like in the OSPIR-OXT protocol of [JJK+13], we ask client $\mathcal{C}$ to reveal the attributes of every term (exact keyword or k-gram) in its query, which allows $\mathcal{D}$ to apply an attribute-based access control policy. To make this policy enforcement effective, we replace PRF keys involved in these OPRF instances with an array of keys, one for each database attribute. Third, to prevent malicious $\mathcal{C}$ from mixing and matching the trapdoors received for different query terms, and thus potentially violate $\mathcal{D}$'s access control policy, we use the same technique as [JJK+13], i.e. $\mathcal{D}$ blinds each trapdoor it obliviously computes, for each term w_i or kg_i, by a random blinding factor ρ_i used in the exponent, e.g. $\mathcal{C}$ computes $\mathsf{xtrap}_i{}^{\rho_i}$ instead of xtrap_i. $\mathcal{D}$ then puts the vector of these ρ_i factors in an authenticated

envelope encrypted under a symmetric key shared by $\mathcal{D}$ and $\mathcal{E}$. During the Search protocol, $\mathcal{E}$ receives this envelope from $\mathcal{C}$, authenticates it, decrypts it, and then adds factor ρ_i^{-1} in the exponent to de-blind the xtag (or ptag) value it computes jointly with $\mathcal{C}$, where $\mathcal{C}$ enters a trapdoor blinded by the corresponding factor ρ_i. In this way ρ_i and ρ_i^{-1} factors cancel each other out in the exponent.

However, while all the above mentioned methods carry over from the OSPIR-OXT protocol of [JJK+13], protocol MIXED-SSE-OXT differs fundamentally from SSE-OXT in one aspect for which we need new techniques. Namely, MIXED-SSE-OXT contains an extra round of interaction in which $\mathcal{C}$ learns the $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}$ values, potentially for each ind encrypted in $\mathbf{T}[w_1]$. Consider two queries $q^{(i)}$ and $q^{(j)}$ whose s-terms are different, i.e. ${w_1}^{(i)} \neq {w_1}^{(j)}$, but their substring queries have the same s-gram i.e. ${\mathsf{kg}_1}^{(i)} = {\mathsf{kg}_1}^{(j)}$. Since $\mathsf{ptag}_{(\mathsf{kg}_1,\mathsf{ind})}$ is a deterministic function of $(\mathsf{kg}_1, \mathsf{ind})$, the ptag values leak the number of common ind's in $\mathsf{DB}({w_1}^{(i)})$ and $\mathsf{DB}({w_1}^{(j)})$, regardless of what information the client $\mathcal{C}$ legitimately gets in $\mathsf{DB}(q^{(i)})$ and $\mathsf{DB}(q^{(j)})$.

We address this problem by modifying the function F_G and the way position information $\mathsf{ind}^{\mathsf{pos}}$ is encrypted in the $\mathbf{P}[(\mathsf{kg}_1, \mathsf{ind})]$ list, which in turn allows us to modify the two-party computation of xtag's $F_G(K_G, (\mathsf{kg}_i, \mathsf{ind}, \mathsf{pos} + \Delta_i))$, for $i = 2, \ldots, \mathsf{h}$, in a way that prevents leakage of any information to $\mathcal{C}$ and at the same time assures that $\mathcal{D}$ learns only the final output of F_G on these inputs. We have several ways of doing this, relying on different computational assumptions and resulting in different pre-computation/on-line efficiency trade-offs. One solution comes from using an elliptic curve group G with a bilinear map $e : G \times G \to G_T$, where F_G can be defined as $F_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}, \mathsf{pos})) = e(\mathsf{xtrap}_{\mathsf{kg}}, h)^{\mathsf{x}\mathsf{ind}^{\mathsf{pos}}}$, and using a variant of ElGamal encryption based on Linear Diffie-Hellman (LDH) assumption on G to jointly compute F_G given the position-related information in $\mathbf{P}$ in the form of encrypted $h^{\mathsf{ind}^{\mathsf{pos}}}$ values.

MIXED-OSPIR-SSE using Bilinear Maps. We provide a more detailed description of the MIXED-OSPIR-SSE variant which uses a group with a bilinear map to allow for a practical two-party computation of xtag's, i.e. of function $F_G((K_X, K_I), (\cdot, \cdot, \cdot))$. Let G be a group of a prime order p with a bilinear map $e : G \times G \to G_T$. Assume that the Linear Diffie-Hellman (LDH) assumption holds on G [BBS04]. Consider function F_G modified as $F_G((K_X, K_I), (\mathsf{kg}, \mathsf{ind}, \mathsf{pos})) = e(\mathsf{xtrap}_{\mathsf{kg}}, h)^{\mathsf{xind}^{\mathsf{pos}}}$, where $\mathsf{xtrap}_{\mathsf{kg}}$ and xind are defined as before, i.e. $\mathsf{xtrap}_{\mathsf{kg}} = H(\mathsf{kg})^{K_X[I(\mathsf{kg})]}$ for H mapping onto G, and $\mathsf{xind} = F_p(K_I, \mathsf{ind})$. We also change the way values $\mathsf{ind}^{\mathsf{pos}}$ are encrypted in list $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$, namely for every pos at which kg appears in $\mathsf{DB}[\mathsf{ind}]$, $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$ contains a *Linear Encryption* (LE) ciphertext $(a, b, c) = \mathsf{Enc}_{(x_1,x_2)}(h^{\mathsf{ind}^{\mathsf{pos}}})$ where h is a generator of G and the encryption key $(x_1, x_2) \in \mathbb{Z}_p \times \mathbb{Z}_p$ is set as $(F_p(\mathsf{strap}_{\mathsf{kg}}, \mathsf{n}_1), F_p(\mathsf{strap}_{\mathsf{kg}}, \mathsf{n}_2))$. The encryption $\mathsf{Enc}_{(x_1,x_2)}(m)$ on message $m \in G$ picks random r, s in $\mathbb{Z}_p$, and outputs $(a, b, c) = (h^s, h^r, m \cdot h^{x_1 \cdot s + x_2 \cdot r})$. The decryption $\mathsf{Dec}_{(x_1,x_2)}(a, b, c)$ outputs $a^{-x_1} \cdot b^{-x_2} \cdot c$.

In the Search protocol, when $\mathcal{E}$ identifies a non-empty list $\mathbf{P}[(\mathsf{kg}, \mathsf{ind})]$, then for each ciphertext $\mathsf{Enc}_{(x_1,x_2)}(h^{\mathsf{ind}^{\mathsf{pos}}})$ in this list, and each x-gram kg_i in the tokenization of $\mathcal{C}$'s substring search term, the two parties perform a sub-protocol whose goal is for $\mathcal{E}$ to compute $\mathsf{xtag}_i = e(\mathsf{xtrap}_{\mathsf{kg}_i}, h)^{\mathsf{xind}^{\mathsf{pos}+\Delta_i}}$. Recall that for each x-gram kg_i, $\mathcal{C}$ holds the shift Δ_i corresponding to kg_i and a blinded trapdoor $(\mathsf{xtrap}_{\mathsf{kg}_i})^{\rho_i}$, while $\mathcal{E}$ holds the corresponding deblinding factor ρ_i^{-1}. Recall also that $\mathcal{C}$ and $\mathcal{E}$ hold a multiplicative sharing, z and y s.t. $z \cdot y = \mathsf{xind}$, hence $z^{\Delta_i} \cdot y^{\Delta_i} = \mathsf{xind}^{\Delta_i}$. Let $B = ((\mathsf{xtrap}_{\mathsf{kg}_i})^{\rho_i})^{z^{\Delta_i}}$, let $v = y^{\Delta_i} \cdot \rho^{-1}$, and denote $h^{\mathsf{ind}^{\mathsf{pos}}}$ encrypted in (a, b, c) as m. The goal of the sub-protocol therefore reduces to computing $e(B, m)^v$, on $\mathcal{E}$'s input $(a, b, c) = \mathsf{Enc}_{(x_1,x_2)}(m)$ and v, and $\mathcal{C}$'s input (x_1, x_2) and B. Note that $e(B, m)^v = e(\mathsf{xtrap}_{\mathsf{kg}_i}, h)^t$ for $t = (\rho_i \cdot z^{\Delta_i}) \cdot \mathsf{ind}^{\mathsf{pos}} \cdot (y^{\Delta_i} \cdot \rho^{-1}) = \mathsf{ind}^{\Delta_i + \mathsf{pos}}$. An additional input into this computation is $\mathcal{E}$'s LE private key (k_1, k_2) and the corresponding public key $(K_1, K_2) = (h^{k_1}, h^{k_2})$ held by $\mathcal{C}$. The computation proceeds as follows:

(1) $\mathcal{E}$ sends the following three tuples to $\mathcal{C}$:

$$(\alpha_a, \beta_a, \gamma_a) \leftarrow \mathsf{Enc}_{(k_1,k_2)}(a)$$
$$(\alpha_b, \beta_b, \gamma_b) \leftarrow \mathsf{Enc}_{(k_1,k_2)}(b)$$
$$(\alpha_c, \beta_c, \gamma_c) \leftarrow \mathsf{Enc}_{(k_1,k_2)}(c)$$

(2) $\mathcal{C}$ picks r_δ, s_δ at random in $\mathbb{Z}_p$, computes

$$\begin{aligned}
\bar{\alpha} &= (\alpha_a)^{-x_1} \cdot (\alpha_b)^{-x_2} \cdot \alpha_c \cdot h^{r_\delta} \\
\bar{\beta} &= (\beta_a)^{-x_1} \cdot (\beta_b)^{-x_2} \cdot \beta_c \cdot h^{s_\delta} \\
\bar{\gamma} &= (\gamma_a)^{-x_1} \cdot (\gamma_b)^{-x_2} \cdot \gamma_c \cdot (K_1)^{r_\delta} \cdot (K_2)^{s_\delta}
\end{aligned}$$

and sends $(\alpha, \beta, \gamma) = (e(B, \bar{\alpha}), e(B, \bar{\beta}), e(B, \bar{\gamma}))$ to $\mathcal{E}$.

(3) $\mathcal{E}$ outputs $\mathsf{xtag} = (\alpha^{-k_1} \cdot \beta^{-k_2} \cdot \gamma)^v [= e(B, m)^v]$.

The crucial point is that when $\mathcal{C}$ uses the decryption key (x_1, x_2) on the twice-encrypted values – first under (x_1, x_2) and then under (k_1, k_2) – then by exponentiation commutativity the result $(\bar{\alpha}, \bar{\beta}, \bar{\gamma})$ is an encryption under key (k_1, k_2) of the same plaintext m which was encrypted under key (x_1, x_2) in (a, b, c). (Terms h^{r_δ}, h^{s_δ}, and $(K_1)^{r_\delta}(K_2)^{s_\delta}$ randomize this re-encryption.)

We note that only the computation of the xtag's corresponding to k-gram positions, i.e. to $(\mathsf{ind}, \mathsf{kg}, \mathsf{pos})$ triples, will be computed using the above approach, while the xtag's corresponding to exact keywords, i.e. to $(\mathsf{ind}, \mathsf{kg})$ pairs, will still be computed as in MIXED-SSE-OXT. This modification does not add new rounds to MIXED-SSE-OXT: Instead of ptag, $\mathcal{E}$ will now

send the three tuples computed as in step (1) above for each pos encrypted in $\mathbf{p} = \mathsf{PSet}[\mathsf{ptag}]$. Moreover, for large databases $\mathcal{E}$ will now have to access the disk to retrieve $\mathbf{p}$ from the disk before it can send its response to $\mathcal{C}$. As for pre-computation, the computational cost increase incurred by this method will be moderate, because the bilinear map operation is done only once per each kg in KG, and the linear encryption operations involve fixed-base exponentiations. However, the on-line procedure cost will be dominated by three pairings per each x-gram kg_i for $i = 2, \ldots, \mathsf{h}$ in the search term and each $\mathsf{ind}^{\mathsf{pos}}$ s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}[\mathsf{kg}_1]$ (and s.t. $\mathsf{ind} \in \mathsf{DB}[w_1]$). We are currently investigating the exact effects of these changes on the overall performance of the protocol.

Security in the OSPIR Setting. The privacy profile of the MIXED-OSPIR-OXT protocol against malicious data owner $\mathcal{D}$, malicious clients $\mathcal{C}$, and honest but curious server $\mathcal{E}$, are very similar to those of the OSPIR-OXT protocol of [JJK+13] for the case of exact-keyword conjunctions. Privacy profile against $\mathcal{D}$ is similar because we use the same mechanisms for adapting our MIXED-SSE-OXT protocol to the OSPIR setting as [JJK+13], namely oblivious computation of the PRF's. However, in addition to revealing the vector of attributes pertaining to the query terms, which enables attribute-based access policy control by $\mathcal{D}$, the MIXED-OSPIR-OXT protocol additionally reveals the number of k-grams in each substring term and their relative positions $\Delta_2, \ldots, \Delta_{\mathsf{h}}$. Formally, if $\mathbf{q}$ is a vector of queries of the form $q = (w_1, \ldots, w_n, q'_1, \ldots, q'_k)$ where each w_i is an exact query term, with w_1 chosen as the s-term (to simplify the presentation), and each q'_i is a substring term s.t. $T(q'_i) = (\mathsf{kg}_{i,1}, (\mathsf{kg}_{i,2}, \Delta_{i,2}), \ldots, (\mathsf{kg}_{i,\mathsf{h}_i}, \Delta_{i,\mathsf{h}_i}))$, then $\mathcal{L}_{\mathcal{D}}(\mathsf{DB}, \mathbf{q})$ consists of DB (since $\mathcal{D}$ is the owner of the database DB) and the following information for each q in $\mathbf{q}$: a vector of attributes $(I(w_1), \ldots, I(w_n), I(q'_1), \ldots, I(q'_k))$, and the vectors of shifts $(\Delta_{i,2}, \ldots, \Delta_{i,\mathsf{h}_i})$ for each substring term q'_i in q. We note that if the moderate leakage of information on the substring terms leaked in the Δ vectors is unacceptable, it can be eliminated by a Δ-hiding variant of our protocol.

The privacy profile to the malicious $\mathcal{C}$ is also very similar to the OSPIR-OXT protocol. As in there, the client learns the size of the TSet list for the s-term keyword or k-gram in the search query. However, since we handle position-related information by the PSet's, one for every substring term in the search query, $\mathcal{C}$ also learns the sizes of these PSet's. Formally, if $\mathbf{q}$ is a vector of queries of the form $q = (w_1, \ldots, w_n, q'_1, \ldots, q'_k)$ where each w_i is an exact query terms, and w_1 is the s-term, and each q'_i is a substring term, and if $T(q'_i) = (\mathsf{kg}_{i,1}, (\mathsf{kg}_{i,2}, \Delta_{i,2}), \ldots, (\mathsf{kg}_{i,\mathsf{h}_i}, \Delta_{i,\mathsf{h}_i}))$, then $\mathcal{L}_{\mathcal{C}}(\mathsf{DB}, \mathbf{q})$ consists of $|\mathsf{DB}(w_1)|$ for s-term w_1 in each query q in $\mathbf{q}$, and $|\mathsf{DB}(\mathsf{kg}_{i,1})|$ for s-term k-gram $\mathsf{kg}_{i,1}$ in each substring term q_i in each query q in $\mathbf{q}$.

The privacy profile to the honest-but-curious server $\mathcal{E}$ is similar as that specified for the SUB-SSE-OXT protocol in Section 6.3.1, but it contains some new elements. For simplicity of notation we will assume that each query has n exact terms and k substring terms, and that each substring search term tokenizes to h k-grams. Building on the above notation, denote the i-th query as $q^{(i)} = ({w_1}^{(i)}, \ldots, {w_n}^{(i)}, {q'_1}^{(i)}, \ldots, {q'_k}^{(i)})$ where ${w_1}^{(i)}$ is an s-term, and let $T({q'_j}^{(i)}) = \left(\mathsf{kg}_{j,1}{}^{(i)}, (\mathsf{kg}_{j,2}^{(i)}, {\Delta_{j,2}}^{(i)}), \ldots, (\mathsf{kg}_{j,\mathsf{h}}{}^{(i)}, {\Delta_{j,\mathsf{h}}}^{(i)})\right)$. Define function $\mathcal{L}_{\mathcal{E}}(\mathsf{DB}, \mathbf{q})$ which specifies leakage to $\mathcal{E}$ as a vector $(N, \overline{\mathbf{s}}, \mathsf{SP}, \mathsf{RP}, \mathsf{DP}, \mathsf{IP}, \mathsf{PSP})$. Leakage elements $N, \overline{\mathbf{s}}, \mathsf{SP}, \mathsf{RP}$ are defined exactly the same as in Section 6.3.1, or indeed as in the underlying OXT protocol of [CJJ+13]. The delta-pattern DP is defined as in Section 6.3.1, except that it is generalized to k substring terms with h k-grams each. (Formally, $\mathsf{DP}[i]$ is the sequence of vectors $\{({\Delta_{j,2}}^{(i)}, \ldots, {\Delta_{j,\mathsf{h}}}^{(i)})\}$ for $j = 2, \ldots, k$.)

The *conditional intersection pattern* IP in $\mathcal{E}$'s leakage function $\mathcal{L}_{\mathcal{E}}$ contains the leakage due to exact keyword terms and the s-grams of the substring terms, which is the same as in the OXT protocol of [CJJ+13], and the leakage due to the remaining k-grams in the substring terms, which is the generalization of the IP leakage described in Section 6.3.1 to the case of multiple substring terms each with multiple k-grams. Formally, we define IP as a tuple $(\mathsf{IPw}, \mathsf{IPs}, \mathsf{IPk})$. The IPw part contains the leakage due to the exact keyword x-terms, exactly

as in the OXT protocol of [CJJ+13]. The IPs part contains the leakage due to the s-grams, i.e. the s-term k-grams in each substring term, which is similar to the leakage IPw because in the MIXED-OSPIR-OXT protocol $\mathcal{E}$ computes a ptag for each s-gram in the same way as it computes an xtag for each exact keyword x-term, namely as a PRF of the (keyword,record-index) pair, and thus both have the same value whenever the (keyword,record-index) pair repeats. Formally, IPw is a $Q \times Q \times n \times n$ table (where Q is the number of queries) where $\mathsf{IPw}[i_1, i_2, j_1, j_2]$ is non-zero only if $i_1 \neq i_2$, i.e. if this entry relates to two different queries, and if ${w_{j_1}}^{(i_1)} = {w_{j_2}}^{(i_2)}$ and $2 \leq j_1, j_2 \leq n$, i.e. if the j_1-th keyword in i_1-th query is the same as the j_2-th keyword in the i_2-th query (with both keywords being x-terms), in which case $\mathsf{IPw}[i_1, i_2, j_1, j_2]$ contains all indexes ind in $\mathsf{DB}({w_1}^{(i_1)}) \cap \mathsf{DB}({w_1}^{(i_2)})$, i.e. indexes of records that contain the s-terms of both i_1-th and i_2-th queries. IPs is a $Q \times Q \times k \times k$ table where $\mathsf{IPs}[i_1, i_2, j_1, j_2]$ is non-zero only if $i_1 \neq i_2$ and ${\mathsf{kg}_{j_1,1}}^{(i_1)} = {\mathsf{kg}_{j_2,1}}^{(i_2)}$, i.e. if the s-gram in the j_1-th substring term in i_1-th query is the same as the s-gram in the j_2-th substring term in the i_2-th query, in which case $\mathsf{IPs}[i_1, i_2, j_1, j_2]$ contains all indexes ind in $\mathsf{DB}({w_1}^{(i_1)}) \cap \mathsf{DB}({w_1}^{(i_2)})$, exactly as in the case of IPw leakge above. The third part of IP leakage is IPk, the leakage due to $\mathcal{E}$'s computation of xtag's for each (k-gram,position,record-index) tuple. This leakage is exactly the same as the IP leakage in the SUB-SSE-OXT protocol described in Section 6.3.1, but generalized to multiple substring terms and k-grams. Formally, IPk is a $Q \times Q \times k \times k \times \mathsf{h} \times \mathsf{h}$ table, where $\mathsf{IPk}[i_1, i_2, j_1, j_2, \ell_1, \ell_j]$ is non-zero only if $i_1 \neq i_2$ and ${\mathsf{kg}_{j_1,\ell_1}}^{(i_1)} = {\mathsf{kg}_{j_2,\ell_2}}^{(i_2)}$, i.e. if the ℓ_1-th k-gram in the j_1-th substring term in i_1-th query is the same as the ℓ_2-th k-gram in the j_2-th substring term in the i_2-th query, in which case $\mathsf{IP}[i_1, i_2, j_1, j_2, \ell_1, \ell_j]$ contains the set of all triples $(\mathsf{ind}, \mathsf{pos}_1, \mathsf{pos}_2)$ (possibly empty) s.t. $(\mathsf{ind}, \mathsf{pos}_1) \in \mathsf{DB}({w_1}^{(i_1)})$, $(\mathsf{ind}, \mathsf{pos}_2) \in \mathsf{DB}({w_1}^{(i_2)})$, and $\mathsf{pos}_2 = \mathsf{pos}_1 + ({\Delta_{j_1,\ell_1}}^{(i_1)} - {\Delta_{j_2,\ell_2}}^{(i_2)})$, i.e. indexes of records which contain the s-terms of the i_1-th and the i_2-th queries, at positions whose relative distance matches the difference between the Δ's associated with the above two k-grams in the tokenization of the corresponding queries.

The last component of $\mathcal{E}$'s leakage function is a *PSet size pattern* PSP, which is a $Q \times k$ table whose entry $\mathsf{PSP}[i, j]$ contains a sequence of integers $(s_1, s_2, \ldots, s_m)$ where $m = |\mathsf{DB}({w_1}^{(i)})|$, s.t. s_c is the number of occurrences of ${\mathsf{kg}_{j,1}}^{(i)}$, i.e. the s-gram in the j-th substring term in the i-th query, in record $\mathsf{DB}[\mathsf{ind}_c]$, where ind_c is the c-th record index in $\mathsf{DB}({w_1}^{(i)})$. This leakage comes from the fact that for each s-gram in each query $\mathcal{E}$ retrieves the (possibly empty) PSet associated with $\mathsf{ptag}[c]$ for $c = 1, \ldots, m$, and the size of this PSet reflects the number of occurrences of k-gram ${\mathsf{kg}_{j,1}}^{(i)}$ in the c-th record in $\mathsf{DB}({w_1}^{(i)})$.

Note: We stress that that above formal specification of $\mathcal{E}$'s leakage is in many ways an overstatement. Most importantly, the real information $\mathcal{E}$ learns due to the IP leakage does not contain the indexes (even randomized) of the records which satisfy the conditioned formed by the two queries, but only the fact that the corresponding TSet entries contain the same index ind.

The proof of theorem 6.2 below is very simple, while the proofs of theorem 6.3 and 6.4, although more complex, are similar to the proof of security against the client and the server of the OSPIR-OXT protocol of [JJK+13]. All proofs are omitted.

Theorem 6.2. *Protocol MIXED-OSPIR-OXT is $\mathcal{L}_{\mathcal{D}}$-semantically-secure against malicious data owner $\mathcal{D}$.*

Theorem 6.3. *Protocol MIXED-OSPIR-OXT is $\mathcal{L}_{\mathcal{C}}$-semantically-secure against a malicious client $\mathcal{C}$, assuming the security of the encryption* Enc*, the authenticated encryption, the* TSet *implementation, the PRF's F_p and F_τ, assuming the random oracle model for hash functions, the One-More GDH and the LDH assumptions on the group G with a bilinear map, the q-DDH assumption on its target group G_T, and the One-More GDH assumption on a standard prime-order group.*

Theorem 6.4. *Protocol MIXED-OSPIR-OXT is $\mathcal{L}_{\mathcal{E}}$-semantically-secure against honest-but-curious server $\mathcal{E}$, assuming the security of the encryption* Enc*, the* TSet *implementation, the*

PRF's F_p and F_τ, the random oracle model for hash functions, and the LDH assumptions on group G.

6.6 Security Proof for Substring Search SSE

Here we present the proof of Theorem 6.1 stated in Section 6.3, which describes the security property of protocol SUB-SSE-OXT, the basic substring search SSE protocol shown in Figure 6.1. We simplify notation by focusing on substrings which tokenize into two k-grams. The extension to any number of k-grams is straightforward.

Hardness assumptions. We recall the q-DDH assumption (we assume familiarity with the DDH assumption). Let G be a prime order cyclic group of order p generated by g. We say that the *q-decision Diffie-Hellman (q-DDH) assumption* holds in G if $\mathcal{A}\lceil\sqsubseteq^{\mathrm{q-ddh}}_{G,A}$ is negligible for any generator g and all efficient adversaries A,

$$\begin{aligned}\mathcal{A}\lceil\sqsubseteq^{\mathrm{q-ddh}}_{G,A} \;=\; & \Pr[A(g, g^a, g^{a^2}, \ldots, g^{a^{q-1}}, g^{a^q}) = 1] \\ & - \Pr[A(g, g^a, g^{a^2}, \ldots, g^{a^{q-1}}, g^b) = 1]\end{aligned}$$

where the probability is over the randomness of A and uniformly chosen a, b from ${\mathbb{Z}_p}^*$.

Note that the q-DDH assumption implies the DDH assumption. We will use the following lemma in the argument below. Let α, β be integers, let $\mathbf{a} \in ({\mathbb{Z}_p}^*)^\alpha$, $\mathbf{b} \in ({\mathbb{Z}_p}^*)^\beta$, and let $\mathbf{q} = (1, 2, \ldots, q)$. Let $\mathbf{a} \cdot \mathbf{b^q}$ be the $(\alpha \times \beta \times q)$ array M s.t. $\mathrm{M}[i, j, k] = \mathbf{a}[i] \cdot \mathbf{b}[j]^k$, and let $g^{\mathbf{a} \cdot \mathbf{b^q}}$ be the $(\alpha \times \beta \times q)$ array M_G s.t. $\mathrm{M}_G[i, j, k] = g^{\mathrm{M}[i,j,k]}$ where $\mathrm{M} = \mathbf{a} \cdot \mathbf{b^q}$.

Lemma 6.1. *If the q-DDH assumption holds in G then for any integers α, β (polynomial in $|p|$) and any efficient adversary A, we have that $\mathcal{A}\lceil\sqsubseteq_G^A$ is negligble, where*

$$\mathcal{A}\lceil\sqsubseteq_G^A = \Pr[A(g, g^{\mathbf{a}\cdot\mathbf{b}^{\mathbf{q}}}) = 1] - \Pr[A(g, \mathrm{M}_G) = 1]$$

where $\mathbf{a}$ *is uniform over* $(\mathbb{Z}_p{}^*)^\alpha$, $\mathbf{b}$ *is uniform over* $(\mathbb{Z}_p{}^*)^\beta$, *and* M_G *is uniform over* $G^{\alpha\times\beta\times q}$.

Let K, X, Y be sets, and let $F : K \times X \to Y$ be a family of keyed functions. We say that F is a *pseudorandom function (PRF)* if for all efficient adversaries A, $\mathcal{A}\lceil\sqsubseteq_{F,A}^{\mathrm{prf}}$ is negligible, where

$$\mathcal{A}\lceil\sqsubseteq_{F,A}^{\mathrm{prf}} = \Pr[A^{F(k,\cdot)} = 1] - \Pr[A^{f(\cdot)} = 1]$$

where the probability is over the randomness of A, $k \xleftarrow{\$} K$, and $f \xleftarrow{\$} \mathsf{Fun}(X, Y)$.

As a corollary of lemma 6.1 we get the following, where $[q]$ stands for the set of integers $\{1, \ldots, q\}$:

Corollary 6.4.1. *If the q-DDH assumption holds in G, if $F_G : K_1 \times X \to G$ and $F_p : K_2 \times Y \to \mathbb{Z}_p{}^*$ are PRF's then $F : (K_1 \times K_2) \times (X \times Y \times [q]) \to G$ where $F((k_1, k_2)(x, y, i)) = (F_G(k_1, x))^{(F_p(k_2,y))^i}$ is a non-adaptive PRF.*

We will also use the following well-known fact:

Lemma 6.2. *Under the DDH assumption on G, for any set X, if H is a hash function mapping X to G then under the DDH assumption function $F : \mathbb{Z}_p{}^* \times X \to G$ defined as $F(k, x) = H(x)^k$ for k uniform in $\mathbb{Z}_p{}^*$, is a PRF in the random oracle model (ROM) for H.*

Proof of Theorem 6.1 from Section 6.3. Let DB be any text strings database and $\mathbf{q}$ be any sequence of Q queries to it as described above. Let $(N, \bar{\mathbf{s}}, \mathsf{SP}, \mathsf{DP}, \mathsf{RP}, \mathsf{IP}) \leftarrow$

$\mathcal{L}_{\mathcal{E}}(\mathsf{DB}, \mathbf{q})$ (note that algorithm $\mathcal{L}_{\mathcal{E}}$ is deterministic). Let A be an efficient algorithm which plays the role of $\mathcal{E}$, i.e. receives the $(\mathsf{TSet}, \mathsf{XSet})$ input generated by $\mathsf{Setup}(\mathsf{DB})$ and input $(\mathsf{stag}, \Delta_1, \mathsf{xtoken}[1], \mathsf{xtoken}[2], \ldots)$ generated by the client $\mathcal{C}$ on input $\mathbf{q}[\tau]$ (and $K = (K_S, K_X, K_T)$ generated in the same $\mathsf{Setup}(\mathsf{DB})$ procedure). We will first make several modifications in the way we look at this information, at some point involving the simulator SIM_T for the underlying T-set implementation, arguing that A's view remains indistinguishable between each consecutive modification. Finally we show a simulator which generates this modified view given only input $(N, \bar{\mathbf{s}}, \mathsf{SP}, \mathsf{DP}, \mathsf{RP}, \mathsf{IP})$, which will complete the proof.

(1) First, we replace $\mathsf{strap} = H(w)^s$ values generated in Setup and $\mathsf{GenToken}$ with random elements in group G. This modification results in an indistinguishable change in A's view because by Lemma 6.2, $H(w)^s$ is a PRF (since q-DDH implies DDH), and key s is never exposed to A. We will denote strap and stag values generated for keyword w_1 as strap_{w_1} and stag_{w_1}.

(2) Secondly, we replace each (K_z, K_e, K_u) triple generated for a given strap_{w_1} with random τ-bit strings. This modification results in an indistinguishable change in A's view because F_τ is a PRF and strap_{w_1} values are never exposed to A. We will denote the key triple generated for a particular strap_{w_1} as $(K_{w_1,z}, K_{w_1,e}, K_{w_1,u})$.

(3) Third, we replace each (z_c, u_c) pair generated for a given $(K_{w_1,z}, K_{w_1,u})$ key and counter c, with random values in ${\mathbb{Z}_p}^*$. This modification results in an indistinguishable change in A's view because $(K_{w_1,z}, K_{w_1,u})$ keys are not exposed to A, and F_p is a PRF. Let us denote the tuple (z_c, u_c) generated for counter c from keys $(K_{w_1,z}, K_{w_1,u})$ as $(z_{w_1,c}, u_{w_1,c})$.

(4) Next, we replace generation of ciphertext e in a $(e, y, [u])$ tuple with $\mathsf{e} \leftarrow \mathsf{Enc}(K_{w_1,e}(0^{2\lambda}))$ instead of $\mathsf{Enc}(K_{w_1,e}, (\mathsf{ind}|\mathsf{rdk}))$ (since ind and rdk are both λ-long bit strings). This modification results in an indistinguishable change in A's view because $(\mathsf{Enc}, \mathsf{Dec})$ is a CPA secure encryption, and the keys $K_{w_1,e}$ are never exposed to A. We will denote the tuple (e, y, u) gen-

erated for s-term w_1 and counter c as $(\mathsf{e}_{w_1,c}, y_{w_1,c}, u_{w_1,c})$. We will also designate $\mathsf{ind}, \mathsf{pos}, \mathsf{xind}$ values corresponding to the c-th position in $\mathbf{T}(\mathsf{stag}_{w_1})$ as $\mathsf{ind}_{w_1,c}, \mathsf{pos}_{w_1,c}, \mathsf{xind}_{w_1,c}$.

For convenience of notation, we will denote the pair of keys (K_X, K_I) as K_{XI}. Define function $F_{\mathsf{xtag}} : (((\mathbb{Z}_p{}^*)^m \times \{0,1\}^\lambda) \times (\{0,1\}^\lambda \times \{0,1\}^\lambda \times [q])) \rightarrow G$ as $F_{\mathsf{xtag}}(K_{XI}, (w, \mathsf{ind}, \mathsf{pos})) = (F_G(K_X, w))^{(F_p(K_I, \mathsf{ind}))^{\mathsf{pos}}}$. Note that XSet consists of values $F_{\mathsf{xtag}}(K_{XI}, (w, \mathsf{ind}, \mathsf{pos}))$ computed for every $(w, \mathsf{ind}, \mathsf{pos})$ tuple s.t. $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(w)$. Observe that values $\mathsf{xtoken}_{\mathsf{kg}}[c, i]$ sent by $\mathcal{C}$ in the Search are of the form

$$\mathsf{xtoken}_{\mathsf{kg}}[c, i] = \mathsf{xtag}^{\left((y_{w_1,c})^{(\Delta_i)} \cdot v_{w_1,c}\right)^{-1}} \tag{6.3}$$

for $\mathsf{xtag} = F_{\mathsf{xtag}}(K_{XI}, (w_i, \mathsf{ind}_{w_1,c}, \mathsf{pos}_{w_1,c} + \Delta_i))$.

(5) Since the value satisfying this equation is unique, we will have Charlie generate the $\mathsf{xtoken}_{\mathsf{kg}}[c, i]$ values by equation (6.3).

(6) The next modification is that we change the way Setup generates $y_{w_1,c}$ and $v_{w_1,c}$ elements in each $\mathbf{T}[\mathsf{stag}_{w_1}]$ tuple, by choosing both of them at random in $\mathbb{Z}_p{}^*$, and then defining $z_{w_1,c}$ and $u_{w_1,c}$ generated by $\mathcal{C}$ in Search as $\mathsf{xind}/y_{w_1,c}$ and $\mathsf{xind}^{\mathsf{pos}}/v_{w_1,c}$, respectively. This modification does not change A's view because either way $y_{w_1,c}, v_{w_1,c}$ are random elements in $\mathbb{Z}_p{}^*$. Note that after the above modification $\mathsf{xtoken}_{\mathsf{kg}}[c, i]$ elements $\mathcal{C}$ sends depend only on $y_{w_1,c}, v_{w_1,c}$, and no longer on $z_{w_1,c}, u_{w_1,c}$.

(7) Therefore, after this modification $\mathcal{C}$ will skip generating $z_{w_1,c}, u_{w_1,c}$ in the Search procedure. In fact, these values will not be generated anywhere in the game.

Let us look closer now at where the Setup procedure, at this point in our series of modifications, needs the key $K_{XI} = (K_X, K_I)$ and the $\mathsf{ind}, \mathsf{xind}, \mathsf{pos}$ values. Note that Setup no longer uses ind, xind, and $\mathsf{xind}^{\mathsf{pos}}$ to generate the $(\mathsf{e}_{w_1,c}, y_{w_1,c}, v_{w_1,c})$ tuples in $\mathbf{T}[\mathsf{stag}_{w_1}]$, and therefore in particular it does not use the K_I key at this point either. The only place

key $K_{XI} = (K_X, K_I)$ (and value xind) is used is in the generation of the xtag value inserted into XSet, i.e. in the generation of $F_{\mathsf{xtag}}(K_{XI}, (w, \mathsf{ind}, \mathsf{pos}))$. Note that by Lemma 6.2, function $F_G : {\mathbb{Z}_p}^* \times \{0,1\}^\lambda \to G$ where $F_G(K_X, w) = H(w)^{e_i}$ is a PRF, for $i = I(w)$ and $K_X = (e_1, \ldots, e_m)$ is chosen uniformly in $({\mathbb{Z}_p}^*)^m$. Therefore, by Corollary 6.4.1, assuming q-DDH, ROM, and the fact that F_p is a PRF, function F_{xtag} is a PRF.

(8) Consequently, in the next modification we replace $F_{\mathsf{xtag}}(K_{XI}, \cdot)$ with a random function $F_{\mathsf{xtag}}(\cdot)$, which assigns a random element in G to every $(w, \mathsf{ind}, \mathsf{pos})$ triple. Since, as we discussed above, key $K_{XI} = (K_X, K_I)$ is not used anywhere else at this point except in the computation of F_{xtag}, and A's view can be generated using black-box access to F_{xtag}, this modification results in an indistinguishable change in A's view.

Let us reassess what A's view consists of at this point. First, recall that by Lemma 6.2, function $F_G : {\mathbb{Z}_p}^* \times \{0,1\}^\lambda \to G$ where $F_G(K_T, w) = H(w)^{k_i}$ is a PRF, for $i = I(w)$ and $K_T = (k_1, \ldots, k_m)$ is chosen uniformly in $({\mathbb{Z}_p}^*)^m$. Using this notation, T-set tuples (e, y, v) are formed as an encryption of $0^{2\lambda}$ (e) and random ${\mathbb{Z}_p}^*$ nonces (y and v), and inserted into TSet with an stag handle computed as $\mathsf{stag}_w = F_G(K_T, w)$, while X-set is populated with values of $F_{\mathsf{xtag}}(\mathsf{ind}, w, \mathsf{pos})$ for all $w \in \mathsf{KG}$ and all $(\mathsf{ind}, \mathsf{pos}) \in \mathsf{DB}(w)$. Then, for each query $\mathbf{q}[i]$, tokenized as $(\mathbf{s}[i], (\mathbf{x}[i], \boldsymbol{\Delta}[i]))$, procedure Search sends to A a singleton $(\Delta_2[i])$ (since we assume that each query tokenizes into two k-grams, we have $\mathsf{h} = 2$), value $\mathsf{stag}_{\mathbf{S}[i]} = F_G(K_T, \mathbf{s}[i])$, and a stream of $\mathsf{xtoken}_{\mathsf{kg}}[c, 2]$ values (again, recall that $\mathsf{h} = 2$) for $c = 1, 2, \ldots$. In the i-th query we will denote these values as $\mathsf{xtoken}_{\mathsf{kg}}[c, 2][i]$. These values are computed as in equation (6.3), but modified by replacing $F_{\mathsf{xtag}}(K_{XI}, \cdot)$ with $F_{\mathsf{xtag}}(\cdot)$, and with (w_1, w_2, Δ_2) terms set to the corresponding values in query $\mathbf{q}[i]$, i.e., they are computed as:

$$\mathsf{xtoken}_{\mathsf{kg}}[c, 2][i] = \mathsf{xtag}^{\left((y_{\mathbf{S}[i],c})^{(\boldsymbol{\Delta}_{[i]})} \cdot v_{\mathbf{S}[i],c}\right)^{-1}} \tag{6.4}$$

for $\mathsf{xtag} = F_{\mathsf{xtag}}(K_{XI}, (\mathbf{x}[i], \mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c} + \mathbf{\Delta}[i]))$.

Therefore, since function $F_G(K_T, \cdot)$ is a PRF and the T-set implementation is secure, in the next change **(9)** we will use simulator SIM_T instead of the T-set implementation. In other words, we compute $\mathbf{T}[\mathbf{s}[i]]$ for each $i = 1, \ldots, Q$ as the set of $\mathsf{SP}[i] = |\mathsf{DB}(\mathbf{s}[i])|$ triples (e, y, v) computed as above, and we run $\mathsf{SIM}_T(N, \mathbf{T})$ to generate the TSet datastructure and the search handles $\mathsf{stag}_{\mathbf{S}[i]}$ for $i = 1, \ldots, Q$. By the security of the T-set implementation, this modification results in an indistinguishable change in A's view.

(10) Finally, we change the generation of the xtag values in XSet and the $\mathsf{xtoken}_{\mathsf{kg}}$ generated in Search as follows: To generate xtag's in XSet we simply chose N random elements in G. We also keep a table XT indexed by $(w_2, \mathsf{ind}, \mathsf{pos})$ triples to which we assign some elements in G as the game progresses. Then, we generate $\mathsf{xtoken}_{\mathsf{kg}}[c, 2][i]$ as follows.

1. First we check if $XT(\mathbf{x}[i], \mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c} + \mathbf{\Delta}[i])$ is already defined. If it is, we move to the second step, but if it is not then we first define this entry in the XT table as follows:

   (a) If $(\mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c})$ is in $\mathsf{DB}(w)$ then we assign to this entry in the XT table to a random un-used value in XSet (i.e. to a random value which is not yet assigned to any other entry in the XT table).

   (b) Otherwise, i.e. if $(\mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c})$ is not in $\mathsf{DB}(w)$ then we assign a random element in G to this entry in the XT table.

2. Secondly, we take $\mathsf{xtag} \leftarrow XT(\mathbf{x}[i], \mathsf{ind}_{\mathbf{S}[i],c}, \mathsf{pos}_{\mathbf{S}[i],c} + \mathbf{\Delta}[i])$ and we compute $\mathsf{xtoken}_{\mathsf{kg}}[c, 2][i]$ as xtag exponentiated to $\left((y_{\mathbf{S}[i],c})^{(\mathbf{\Delta}[i])} \cdot v_{\mathbf{S}[i],c}\right)^{-1}$.

It follows by the randomness of F_{xtag} and by equation (6.4) that the above modification does not change A's view: The xtag values remain random and the only thing that A can observe

is whether or not the xtag's computed for each $[c, 2, i]$ hit some previously observed xtag value, and whether this value is in XSet or not.

We will argue that the above view can be generated given only leakage $(N, \bar{\mathbf{s}}, \mathsf{SP}, \mathsf{DP}, \mathsf{RP}, \mathsf{IP})$ generated by $(\mathsf{DB}, \mathbf{q})$. By the security of the T-set implementation A's views of TSet, of the $\mathsf{stag}_{\mathbf{S}[i]}$ values, and hence also of the $\mathbf{T}[\mathbf{s}[i]]$ vectors of (e, y, v) tuples retrieved from TSet via $\mathsf{stag}_{\mathbf{S}[i]}$'s, is simulated correctly on input $(N, \bar{\mathbf{s}}, \mathsf{SP})$. Leakage $\mathsf{DP}[i]$ is used directly as $\boldsymbol{\Delta}[i]$. The result pattern $\mathsf{RP}[i]$ is used to assign some random positions c for each $\mathbf{T}[\mathbf{s}[i]]$ to the $(\mathsf{ind}, \mathsf{pos})$ values in $\mathsf{RP}[i]$, and to decide whether the xtag computed for this position should be from XSet or not. Finally, the IP leakage is used to detect repetitions in the xtag values, i.e. to simulate the view from step (10) above without the XT table. (Note that the XT table keeps all the xtag's which A sees/computes during Search not only for values $(\mathbf{x}[i], \mathsf{ind}, \mathsf{pos}+\Delta)$ corresponding to $(\mathsf{ind}, \mathsf{pos}+\Delta)$ in $\mathsf{DB}(\mathbf{x}[i])$ and $(\mathsf{ind}, \mathsf{pos})$ in $\mathsf{DB}(\mathbf{s}[i])$, which the simulator can simulate using $\mathsf{RP}[i]$, but also for values which are not in DB. Here is where IP table is necessary: For every $\mathbf{q}[i], \mathbf{q}[j]$ pair, IP gives to the simulator the set of ind's with the corresponding positions $\mathsf{pos}_i, \mathsf{pos}_j$, s.t. $\mathbf{x}[i] = \mathbf{x}[j]$ and $\mathsf{pos}_i + \boldsymbol{\Delta}[i] = \mathsf{pos}_j + \boldsymbol{\Delta}[j]$, which is precisely the information needed to detect when some xtag value (i.e. some entry in the XT table) should repeat. Since this simulated view is identical to the view in step (10), the theorem follows.

Chapter 7

Related Work

In this chapter, we provide an overview of prior work related to this dissertation. We discuss related work on: GenoDroid, UnLinked and SPH-PSM.

7.1 Privacy Preserving Genomic Testing on Smartphones

This section discusses prior work on security and privacy of genomics, as well as secure computation focusing on mobile devices.

7.1.1 Secure Testing on Fully Sequenced Human Genomes

Non-cryptographic approaches to privacy, such as de-identification, are often ineffective on genomic data, as shown by some recent studies [Mal05, H$^+$08, W$^+$09, ZPL$^+$11]. As a result, several privacy-preserving cryptographic techniques have been proposed. We now review techniques for secure testing on fully sequenced human genomes.

Baldi, et al. [BBD+11] recently introduced several cryptographic protocols for privacy-preserving testing of fully sequenced human genomes, including RFLP-based paternity test and genetic screening for personalized medicine or recessive genetic diseases. Similar to our setting, individuals obtain their genomes and allow authorized parties (e.g., doctors) to run genetic tests such that only test results are disclosed to one or both parties. However, [BBD+11] only addresses the issue of designing cryptographic protocols, and does not deal with real-world issues. In particular: genome conversion, extensibility, fine-grained optimizations on mobile devices and usability, are not considered. Whereas, our work provides a set of working practical instantiations of genomic tests on a popular smartphone platform. Also, in contrast with the PM technique in [BBD+11], our work includes a cloud-aided variant that facilitates much more efficient operation. Finally, we design and implement new operations, such as privacy-preserving genetic ancestry testing.

Chen, et al. [CPWT12] studied the problem of privacy-preserving mapping and aligning of human genomic sequences to a reference genome, by outsourcing work to the cloud and protecting sensitive DNA information. Since [CPWT12] does not consider genomic testing, it is orthogonal to our work.

7.1.2 Secure Computation on DNA Fragments

We now review prior work realizing secure computation on DNA fragments, as opposed to fully sequenced genomes.

Bruekers, et al. [BKKT08] presented privacy-preserving techniques for some DNA operations, based on Short Tandem Repeat (STR). Proposed techniques use homomorphic encryption on DNA fragments to perform comparisons. Testing protocols are resilient to small numbers of errors, however, their complexity increases with the number of tolerated errors [BA10]. Also, [BKKT08] leaves as an open problem the scenario where an attacker faithfully runs the

protocol but with arbitrary inputs. In this setting, an attacker, given STR's limited entropy, can "lie" about its STR profiles and run multiple dependent protocols, thus reconstructing the other party's profile.

Wang, et al. [WWL+09] developed techniques for computation on genomic data stored at a data provider, including: edit distance, Smith-Waterman and search for homologous genes. Program specialization is used to partition genomic data into "public" (most of the genome) and "sensitive" (a very small subset of the genome). Sensitive regions are replaced with symbols by data providers (DPs) before data consumers (DCs) have access to genomic information.

Troncoso-Pastoriza, et al. [TPKC07] proposed an error-resilient privacy-preserving protocol for string searches. One party, on input of its DNA snippet, can verify the existence of a short template (e.g., a genetic test held by the service provider) within its (short) snippet. This technique handles errors and maintains privacy of both the template and the snippet. Each query is represented as an automaton executed using a finite state machine (FSM) in an oblivious manner. However, the number of FSM states is always revealed to all parties.

The work of Katz, et al. [KM10a] also considers DNA testing. Specifically, it realizes secure computation of the CODIS test [The11] (run by the FBI for DNA identity testing), that could not be otherwise implemented using pattern matching or FSM.

Another set of cryptographic results focus on privately computing the *edit distance* for two strings α, β. (Edit distance is defined as the minimum number of operations, such as, delete, insert, or replace, needed to transform α into β.) Privacy-preserving computation of Smith-Waterman scores [SW81] has also been investigated and used for sequence alignment. Jha, et al. [JKS08] show how to securely compute edit distance using garbled circuits [Yao82], and demonstrate that the resulting overhead is acceptable only for small strings (e.g., a 200-character strings require 2GB circuits). For longer strings, they propose two optimized

techniques, that exploit the structure of the dynamic programming problem (intrinsic to the specific circuit) and split the computation into smaller component circuits. However, a quadratic number of oblivious transfers is needed to evaluate garbled circuits, thus limiting scalability of this approach. Nonetheless, 500-character string instances take almost one hour to complete, according to [JKS08]. Optimized protocols also extend to privacy-preserving Smith-Waterman scores [SW81], a more sophisticated string comparison algorithm, where costs of delete/insert/replace operations, instead of being equal, are determined by special functions. Again, scalability is limited: experiments in [JKS08] show that evaluation of Smith-Waterman for a 60-character string takes about 1,000 seconds.

7.1.3 Secure Computation on Mobile Devices.

In [HCE11], Huang, et al. present a preliminary analysis of the performance of pipelined garbled circuits on smartphones and deploy optimized circuit-based techniques for secure computation [HEKM11]. [HCE11] mentions *personal genetics* as a possible application, showing that two individuals could compare 25 genetic features (e.g., common recessive genetic diseases) in about 7 seconds. Although, in this work, we do not focus on this test, we observe that only if *all* known recessive diseases are compared, the test becomes truly privacy-preserving. (In fact, an individual who requests a specific subset of tests may be revealing the disease he/she suffers). While it may be reasonable today to assume that the number of known recessive diseases is in the order of 25, this may soon become unrealistic, as full genome sequencing continuously enables better understanding of the genome and discovery of new diseases. Also, [HCE11] presents, as an application example, *CommonContacts*, an Android app that privately discovers the contacts common between two users. It realizes Private Set Intersection [FNP04], where sets correspond to contact lists. However, computation overhead appears to be still too high, in practice, to scale up to fully sequenced genomes, since executing *CommonContacts* on lists with 256 entries takes about 10 minutes [HCE11].

Carter, et. al [CADT11] presented the concept of *Efficient Mobile Oblivious Computation* (EMOC), i.e., a technique that completely replaces garbled circuits with homomorphic operations on ciphertexts. EMOC is used to solve Yao's millionaires problem [Yao82] and compute common friends in a social network. Finally, Mood, et al. [MLB12] have recently proposed a memory optimization for garbled-circuit based applications for generic secure computation on mobile phones.

7.1.3.1 Cloud-Aided Secure Computation

With the increasing availability of low-cost, high-performance computing platforms *in the cloud*, the research community has started proposing protocols for secure computation that rely on cloud providers to help increase their efficiency.

Kamara, et. al [KMR11] propose a server-aided setting for Secure Multi-party Computation (SMC) where interacting parties have access to a single server that does not have any input or output to/from the computation but has a vast (yet bounded) amount of computational resources. In this setting, they design protocols that minimize the computation of the parties at the expense of the server. A similar intuition has also been used in previous work, such as, [FKN94, Bea97, Cac99, P$^+$11, DLT11]. However, none of these techniques apply to the setting of human genome testing.

7.2 Private Off-line Social Network Interactions

Most related work falls into two groups: (1) private discovery of common friends and (2) cryptographic protocols for private set operations.

7.2.1 Private Friends Discovery

In von Arb et al. [vABKW08] privacy is considered in the *friend-of-friends* scenario where two users want to learn whether they have the same contacts on their mobile phones, based on phone numbers. The proposed technique, based on [HFH99], uses commutative encryption: each party encrypts its own elements and gets the resulting ciphertexts encrypted by the other party. By comparing encrypted ciphertexts both parties identify common elements. This approach offers no authorization of set elements. Thus, parties can use any phone numbers as protocol input and learn the other party's contacts. Moreover, there are no ETR features, meaning that a malicious party can end the protocol as soon as it learns the intersection.

De Cristofaro et al. [DCMP13] introduced the concept of *private contact discovery* that offers privacy-preserving computation of common contacts. The proposed scheme uses *index-based message encoding* [MPP10]. Unauthorized relaying of contact set elements is addressed via contact certification which requires no trusted third party. The main idea is that each user U issues to its every contact V a contact certificate, which is essentially U's signature on V's id. Thus a certificate is bound by the issuer to a specific user (contact) and can not be relayed. As [vABKW08], this scheme is prone to early protocol termination attacks. Also, it is limited to connections and does not consider other types of profile information, e.g., educational insitutions or past employers, for which such contact certificates are not applicable.

Nagy et al. [NDCD+13] construct a *Common Friends* protocol that allows two parties to privately learn whether they are already friends or share some OSN friends. The protocol uses so-called *bearer capabilities* [TMvR86] for authenticity of friends lists. Bearer capabilities constitute proofs of friendship. Other than containing IDs of OSN users, those capabilities are considered to be "high-entropy objects". Consequently [NDCD+13] claims that using

full-blown (standard) PSI protocols is overkill since set elements are not "predictable". This allows the usage of more efficient PSI protocols based on Bloom filters [Blo70]. Concerning authenticity, [NDCD+13] acknowledges that *re-distribution* of contacts is not addressed.

7.2.2 Private Set Intersection

Private set intersection (PSI) protocols [DCT09], [JL10], [CZ09], [FNP04], [HL08] allow two parties, each with its own set, to privately compute a set intersection. In other words, if two parties' (private) sets include common elements, one or both learn(s) these elements and no information about other set elements (other than their number) is revealed. If both parties learn the result, the protocol is called *mutual* or two-way PSI. It is claimed in [DCT09] that mutual PSI can be obtained by two instantiations of *one-way* PSI. However, we believe that this holds only in the semi-honest model, where protocol executions require no binding. In an *authorized* PSI protocol, set elements need to be signed beforehand by a trusted third party. In this setting, special care must be taken to disallow trading authorized inputs. This is typically not addressed.

7.2.3 Policy-Enhanced Private Set Intersection

Another class of related protocols is called *Policy-Enhanced Private Set Intersection* (PE-PSI) [SSS12]. These allow a ATW-PSI to be constructed from any PSI protocol secure against a malicious adversary. The scheme from [SSS12] offers super-linear computational complexity. Also, binding is not provided: any individually authorized set element can be omitted from the intersection. Furthermore, this scheme requires a PSI protocol secure in the malicious model, which may not be two-way. Thus, in some instantiations, output integrity is not provided. In a practical deployment, such as *UnLinked*, the added costs of such schemes may be prohibitive. In contrast, ATW-PSI and ATW-PSI-CA provide efficient

binding, as well as ETR in the malicious model, and can be optionally extended to provide output integrity.

7.3 Secure Genomics Pattern Matching

Related work falls into several categories described separately below.

7.3.1 Secure Genomics

Motivated by extreme sensitivity of DNA data, the security research community proposed some cryptographic techniques for secure computation on short DNA *fragments*, such as: searching [BKKT08, BA10, TPKC07], computing distance between snippets [WWL+09, JKS08] and related functionalities [FDH+12, KM10a]. With the advent of affordable technologies for WGS, focus shifted to protocols that can scale to the entire genome. In particular, [BBD+11] introduced protocols for paternity testing and personalized medicine. Similar to our work, it is assumed that an individual retains control of (i.e., securely stores) his digitized genome. The protocol for secure personalized medicine testing in [BBD+11] involves (i) a patient, on input her genome, and (ii) a testing facility, on input a list of DNA mutations, along with their corresponding positions. The testing facility needs to check for the presence of these markers in the patient's genome. At the end of the interaction, the patient has no output, while the testing facility learns which markers appear in the patient's genome. The construction in [BBD+11] is based on the Private Set Intersection (PSI) concept [FNP04] and thus has the following limitations:

- If the patient's genome contains only a subset of tested mutations, this subset is revealed to the testing facility, hence violating the goal of only disclosing the test outcome.

- The number of tested markers is revealed to the patient. This information might be enough to (partially or entirely) disclose the nature of the test, thus potentially violating the requirement of hiding test specifics from the patient.

Subsequently, [DCFGT12] extended the work in [BBD+11] by implementing privacy-preserving ancestry (e.g., paternity) testing on Android devices and demonstrating its current viability. However, these results exhibit the same limitations as [BBD+11]. The work in [CKM12] proposed to secure biomedical data using cryptographic hardware, and [KJLM08] used homomorphic encryption to perform scientific investigations on integrated genomic data. Finally, [CPWT12] proposed techniques to securely map and align human genomic sequences to a reference genome, while outsourcing computation to a hybrid cloud.

7.3.2 Secure Pattern Matching

There are a few cryptographic techniques for Secure Pattern Matching (SPM) [HL08, GHS10, HT10, BEM+12], where one party (P_1) holds a pattern and the other party (P_2) holds a text string. P_1 learns where the pattern appears in the text, without revealing it to P_2, or learning anything else about P_2's input. However, the size of P_1's pattern is always revealed to P_2. Although [HT10] sketched out a way to hide the pattern size by means of wildcard padding, the upper bound on the size is still revealed. Plus, supporting wildcards causes a communication and computational performance increase from linear to multiplicative in the size of the text (n) and the size of the pattern (m). In the genomic setting, even a pattern of length 4 would result in around 12 billion modular exponentiations. In order to completely hide the pattern size, communication and computational complexity further increases to $O(n^2)$.

Moreover, SPM actually reveals *all* occurrences of P_1's pattern in P_2's string. Therefore, it is not well-suited for the problem at hand, since, our setting only needs a binary output indicating whether a substring, representing the marker(s) to be tested, appears in a larger string (i.e., the patient's genome) at some specific position(s). It is possible to extend SPM to only disclose the presence of a contiguous substring in a string at a specific location, e.g., by modifying parties' inputs from a sequence of letters to a sequence of hash(letter||position). However, there is no straightforward way to adapt SPM to match *non-contiguous* substrings, except for surrounding the pattern with single-character wild cards, which (as noted earlier) considerably increases protocol complexity.

Based on the above discussion and to the best of our knowledge, no SPM technique can efficiently handle genomic-scale inputs. Moreover, there are scarcely any SPM implementations; one is described [BEM+12]. Whereas, we report on the performance of an actual prototype which confirms the practicality of the proposed SPH-PSM technique.

7.3.3 Input-Size Hiding

There are only a few constructions that support hiding input size in secure computation protocols. Ishai and Paskin [IP07] did so in the generic context of branching programs: one party can evaluate a program on some encrypted input, in such a way that the size of the program is not revealed to the other party. In [ACT11], Ateniese et al. showed that the assumption that secure multi-party computation necessitates revealing input sizes does not always hold. [ACT11] demonstrated a Size-Hiding Private Set Intersection (SHI-PSI) protocol where the size of the set held by the party receiving the intersection (client) is not disclosed.

Although somewhat relevant to the problem at hand, SHI-PSI cannot be used for private substring matching, since, as discussed above, any known linear reduction to PSI leaks

the subset of matching mutations. Note, however, SHI-PSI could be reduced to substring matching (only for contiguous substrings) with *quadratic* computation and communication complexities: the text holder could encode each possible substring as a set element, thus creating a set with n^2 elements; meanwhile, the pattern could be represented as a single element set, and substring matching can be executed as a set intersection. Finally, Lindell et al. [LNO12] recently presented some feasibility results on hiding input size in secure computation, based on Fully Homomorphic Encryption (FHE).

Chapter 8

Conclusions

This dissertation showcased the practicality of Private Set Intersection (PSI) and related variants in solving real world problems.

Chapter 2 explored the viability and practicality of privacy-agile computational genomic tests in the portable and pervasive setting of modern smartphones. We combined domain knowledge in biology, genomics, ubiquitous computing, and applied cryptography, to design and build a personal genomic toolkit called GenoDroid. We implemented it on the Android platform, assessed its performance and conducted a pilot usability study that produced some encouraging results. We have an ongoing effort to incorporate support for additional genetic tests in GenoDroid, broadening its impact. One promising direction is organ donor-recipient genetic compatibility testing. We also intend to look into computational genetic tests for non-human digitized genomes, e.g., plants as well as pets and livestock. Finally, work remains to be done in terms of user perception (and general usability) of personal computational genetic tests and our framework.

In Chapter 3, we reported on the design of *UnLinked* that supports private off-line interaction among nearby OSN users. As part of this work, we developed a novel ATW-PSI protocol that

allows two parties to learn the intersection of their pre-authorized private input sets. Pre-authorization of these input sets by ULS prevents transfer and manipulation of individual set elements. A fully functional prototype of *UnLinked* is available for Android smartphones, allowing LinkedIn users to automatically discover nearby peers who share a sufficient number of friends or other profile features. This discovery happens seamlessly in the background and requires no user interaction until a match is found.

We intend to integrate information from multiple OSNs, e.g. *Facebook*, *Google+* and *Twitter*. This will allow for more flexible policies and would allow us to tap into a greater user pool. Furthermore, we plan to investigate whether our approach is applicable to other application scenarios, such as mobile social services relying on encounter-based trust.

Chapter 5 presented a novel cryptographic primitive called Size- and Position-Hiding Private Substring Matching (SPH-PSM) that appeals to the increasingly relevant scenario of privacy-preserving genomic testing (Personalized Medicine). A prototype implementation attests to the practicality of the proposed technique.

We intend to explore the use of SPH-PSM in the context of unordered sets, i.e., to realize efficient two-party private set disjointness test, while hiding the size of one party's set. Let one party (P_1) represent its set as a polynomial $\mathcal{P}$ (with roots corresponding to set items) and send encrypted coefficients to the other party (P_2). P_2, for each element y_j in its set, can obliviously evaluate $\mathcal{P}$ at y_j using additively homomorphic encryption and compute $ee_j = E(r_j\mathcal{P}(y_j))$ for a random value r_j. After multiplying all ee_j values, P_2 can send the result to P_1 which upon decryption only learns whether the sets are disjoint.

We also intend to distribute an optimized open-source implementation of SPH-PSM along with implementations of some common Personalized Medicine tests such as: *hla-B*, *tpmt*, as well as API support for application developers. This update will be released as a new version of and extension to the GenoDroid framework.

Finally, Chapter 6 presents a significant advance in the ability to run truly complex queries on encrypted data in a variety of operational and trust models. Specifically, we augmented the capabilities of the OXT protocol from the works of [CJJ+13, JJK+13, CJJ+14] to support substring, wildcard and phrase queries, and to allow any combination of these query types under boolean expressions. By leveraging and expanding the underlying machinery of OXT we were able to build on the impressive scalability of the protocol, and while the new query types carry costs in performance and storage, we demonstrated their practicality through a prototype implementation tested under large scale databases by an independent evaluator. One important conclusion is that searching on outsourced encrypted data with significant functionality and privacy-preserving properties is practical today even for large databases. We would like to see a deployment of these technologies in the near future.

Bibliography

[10016] 1000 Genomes Project. A Deep Catalog of Human Genetic Variation. `http://www.1000genomes.org/`, 2016.

[23a16] 23andMe. `https://www.23andme.com/`, 2016.

[Abb03] A. Abbott. Special section on human genetics: With your genes? Take one of these, three times a day. *Nature*, 425(6960), 2003.

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO 2000*, pages 255–270. Springer, 2000.

[ACT11] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (If) size matters: Size-hiding private set intersection. In *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 156–173. Springer, 2011.

[ADHT13] Erman Ayday, Emiliano De Cristofaro, Jean-Pierre Hubaux, and Gene Tsudik. The Chills and Thrills of Whole Genome Sequencing. *To Appear in IEEE Computer*. Available from `http://arxiv.org/abs/1306.1264`, 2013.

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: efficient protocols for realistic adversaries. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 137–156, Berlin, Heidelberg, 2007. Springer-Verlag.

[ATT10] ATT. Mobilizing Enterprise Applications, 2010.

[B+07a] D. Bolnick et al. GENETICS: The Science and Business of Genetic Ancestry Testing. *Science*, 318(5849), 2007.

[B+07b] P. Burton et al. Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. *Nature*, 447, 2007.

[BA10] M. Blanton and M. Aliasgari. Secure outsourcing of dna searching via finite automata. In *DBSec*, 2010.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Christian Cachin and Jan Camenisch,

editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004.

[BBD+11] 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.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology–CRYPTO 2004*, pages 41–55. Springer, 2004.

[Bea97] D. Beaver. Commodity-based cryptography. In *STOC*, 1997.

[BEM+12] Joshua Baron, Karim El Defrawy, Kirill Minkovich, Rafail Ostrovsky, and Eric Tressler. 5PM: Secure pattern matching. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12: 8th International Conference on Security in Communication Networks*, volume 7485 of *Lecture Notes in Computer Science*, pages 222–240, Amalfi, Italy, September 5–7, 2012. Springer, Berlin, Germany.

[BFT16] Tatiana Bradley, Sky Faber, and Gene Tsudik. Bounded size-hiding private set intersection. In *International Conference on Security and Cryptography for Networks*, pages 449–467. Springer, 2016.

[BKKT08] F. Bruekers, S. Katzenbeisser, K. Kursawe, and P. Tuyls. Privacy-Preserving Matching of DNA Profiles. `http://eprint.iacr.org/2008/203`, 2008.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[Blu07] Bluetooth SIG, Simple Pairing Whitepaper. `http://preview.tinyurl.com/bluetooth-simple-pairing`, 2007.

[BPMO12] E. Blass, R. Di Pietro, R. Molva, and M. Onen. PRISM: Privacy-Preserving Searches in MapReduce. In *PETS*, 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[Bro96] J. Brooke. SUS-a quick and dirty usability scale. *Usability evaluation in Industry*, 189, 1996.

[Bro97] A. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences*, 1997.

[Bur12] A. Burke. Foundation Medicine: Personalizing Cancer Drugs. `http://is.gd/foundation_medicine`, 2012.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of cryptography*, pages 535–554. Springer, 2007.

[Cac99] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *CCS*, 1999.

[CADT11] H. Carter, C. Amrutkar, I. Dacosta, and P. Traynor. Efficient Oblivious Computation Techniques for Privacy-Preserving Mobile Applications. Technical report, 2011. `http://smartech.gatech.edu/handle/1853/42367`.

[Can07] T. Canli. The emergence of genomic psychology. *Nature*, 8, 2007.

[Can16] Canalys Research. Smart phones overtake client PCs in 2011. `http://www.canalys.com/newsroom/smart-phones-overtake-client-pcs-2011`, 2016.

[Car08] B. Carlson. SNPs – A shortcut to personalized medicine. *Genetic Engineering & Biotechnology News*, 2008.

[Cas10] S. Cass. Cheap DNA sequencing will drive a revolution in health care. `http://www.technologyreview.com/biomedicine/24587/`, 2010.

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 79–88, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[CGKS95] B. Chor, Oded Goldreich, E. Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*. IEEE, 1995.

[CHLR96] H.R. Chaikind, J. Hearne, B. Lyke, and S. Redhead. The health insurance portability and accountability act (hipaa) of 1996: Overview and guidance on frequently asked questions. 1996.

[CJJ+13] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.

[CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very large databases: Data structures and implementation. In *Symposium on Network and Distributed Systems Security (NDSS 2014)*, 2014.

[CK10] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594, Singapore, December 5–9, 2010. Springer, Berlin, Germany.

[CKM12] M. Canim, M. Kantarcioglu, and B. Malin. Secure Management of Biomedical Data With Cryptographic Hardware. *IEEE Transactions on Information Technology in Biomedicine*, 16(1), 2012.

[CL01] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. *EUROCRYPT*, 2001.

[CM01] F. Collins and V. McKusick. Implications of the Human Genome Project for medical science. *Jama*, 285(5), 2001.

[CM05] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *ACNS 05: 3rd International Conference on Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455, New York, NY, USA, June 7–10, 2005. Springer, Berlin, Germany.

[CMS09] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook : a privacy preserving online social network leveraging on real-life trust. *"IEEE Communications Magazine", Vol 47, N12*, 12 2009.

[Col99] F. Collins. Medical and societal consequences of the human genome project. *New England Journal of Medicine*, 341(1), 1999.

[CPWT12] Y. Chen, B. Peng, X. Wang, and H. Tang. Large-Scale Privacy-Preserving Mapping of Human Genomic Sequences on Hybrid Clouds. In *NDSS*, 2012.

[CS14] Melissa Chase and Emily Shen. Pattern matching encryption. Cryptology ePrint Archive, Report 2014/638, 2014. `http://eprint.iacr.org/`.

[CZ09] Jan Camenisch and GregoryM. Zaverucha. Private intersection of certified sets. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 108–127. Springer Berlin Heidelberg, 2009.

[D. 16] D. Barnett. BamTools. `https://github.com/pezmaster31/bamtools`, 2016.

[DCFGT12] Emiliano De Cristofaro, Sky Faber, Paolo Gasti, and Gene Tsudik. Genodroid: are privacy-preserving genomic tests ready for prime time? In *WPES*, pages 97–108. ACM, 2012.

[DCFT13] Emiliano De Cristofaro, Sky Faber, and Gene Tsudik. Secure genomic testing with size- and position-hiding private substring matching. In *WPES*, pages 107–118. ACM, 2013.

[DCKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *Advances in Cryptology-ASIACRYPT 2010*, pages 213–231. Springer, 2010.

[DCMP13] Emiliano De Cristofaro, Mark Manulis, and Bertram Poettering. Private discovery of common social contacts. *International Journal of Information Security*, 12(1):49–65, 2013.

[DCT09] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear computational and bandwidth complexity. *IACR Cryptology ePrint Archive*, 2009:491, 2009.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO*, 1989.

[DGT12] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, pages 218–231, 2012.

[Dia13] Diaspora Foundation. Webpage, 2013. `https://diasporafoundation.org/`.

[DJ01] Ivan Damgrard and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *PKC*, 2001.

[DLT11] E. De Cristofaro, Y Lu, and G. Tsudik. Efficient techniques for privacy-preserving sharing of sensitive information. In *Trust*, 2011.

[DT10] E. De Cristofaro and G. Tsudik. Practical Private Set Intersection Protocols with Linear Complexity. In *FC*, 2010.

[DT11] E. De Cristofaro and G. Tsudik. Fast and Private Computation of Set Intersection Cardinality. Cryptology ePrint Archive, 2011.

[EDF13] Karim El Defrawy and Sky Faber. Blindfolded data search via secure pattern matching. *Computer*, 46(12):68–75, 2013.

[ElG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on Information Theory*, 31(4), 1985.

[End89] D. Endean. RFLP analysis for paternity testing: observations and caveats. In *Human Identification*, 1989.

[F+09] M. Fumagalli et al. Parasites represent a major selective force for interleukin genes and shape the genetic predisposition to autoimmune conditions. *Experimental Medicine*, 206(6), 2009.

[FDH+12] M. Franz, B. Deiseroth, K. Hamacher, S. Jha, S. Katzenbeisser, and H. Schröder. Towards secure bioinformatics services (short paper). *Financial Cryptography and Data Security*, 2012.

[FJK+15] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In *Proceedings of the Twentieth European Symposium on Research in Computer*

Security (ESORICS), volume 9327 of *Lecture Notes in Computer Science*, pages 123–145. Springer-Verlag, Berlin Germany, 2015. Part II.

[FJKW15] Sky Faber, Stanislaw Jarecki, Sotirios Kentros, and Boyang Wei. Three-party oram for secure computation. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 360–385. Springer, 2015.

[FKN94] U. Feige, J Killian, and M. Naor. A minimal model for secure computation. In *STOC*, 1994.

[FNP04] MichaelJ. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and JanL. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin Heidelberg, 2004.

[FP01] Pierre-Alain Fouque and David Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In *ASIACRYPT*, 2001.

[FPT15] Sky Faber, Ronald Petrlic, and Gene Tsudik. Unlinked: Private proximity-based off-line osn interaction. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, pages 121–131. ACM, 2015.

[Fre05] David Freeman. Pairing-based identification schemes. *arXiv preprint cs/0509056*, 2005.

[FSC11] J. Fowler, J. Settle, and N. Christakis. Correlated genotypes in friendship networks. *Proceedings of the National Academy of Sciences*, 108(5), 2011.

[Gen12a] Genomics Law Report. Patenting and Personal Genomics: 23andMe Receives its First Patent, and Plenty of Questions. `http://preview.tinyurl.com/7ebpft9`, 2012.

[Gen12b] Genomics Law Report. Some Thoughts on Myriad After the Supreme Court Argument. `http://preview.tinyurl.com/bqy25wz`, 2012.

[GHS10] R. Gennaro, C. Hazay, and J. Sorensen. Text Search Protocols with Simulation Based Security. In *PKC*, 2010.

[GMG+13] Melissa Gymrek, Amy L McGuire, David Golan, Eran Halperin, and Yaniv Erlich. Identifying personal genomes by surname inference. *Science*, 339(6117):321–324, 2013.

[gmp16] The gnu multiple precision arithmetic library. `http://gmplib.org/`, 2016.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[Goh03] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/.

[Gol04] Oded Goldreich. *The Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2004.

[Goo16] Google. Dalvik. http://code.google.com/p/dalvik/, 2016.

[GSMG11] D. Greenbaum, A. Sboner, X.J. Mu, and M. Gerstein. Genomics and privacy: Implications of the new reality of closed data for the field. *PLoS Computational Biology*, 7(12), 2011.

[Gut96] P Gutmann. Secure Deletion of Data from Magnetic and Solid-state Memory. In *Usenix Security*, 1996.

[GW09] G. Ginsburg and H. Willard. Genomic and personalized medicine: foundations and applications. *Translational Research*, 154(6), 2009.

[H+08] N. Homer et al. Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays. *PLoS Genetics*, 4(8), 2008.

[Har12] D. Hardt. The OAuth 2.0 authorization framework, Oct. 2012.

[HCE11] Y. Huang, P Chapman, and D. Evans. Privacy-preserving applications on smartphones. In *HotSec*, 2011.

[HEKM11] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Usenix Security*, 2011.

[HFH99] Bernardo A. Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *In Proc. of the 1st ACM Conference on Electronic Commerce*, pages 78–86. ACM Press, 1999.

[HG09] L.E. Hood and D.J. Galas. P4 Medicine: Personalized, Predictive, Preventive, Participatory A Change of View that Changes Everything. http://www.cra.org/ccc/docs/init/P4_Medicine.pdf, 2009.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography*, pages 155–175. Springer, 2008.

[Hof07] M. Hoffman. The genome-enabled electronic medical record. *Journal of Biomedical Informatics*, 40(1), 2007.

[HT10] Carmit Hazay and Tomas Toft. Computationally secure pattern matching in the presence of malicious adversaries. In *ASIACRYPT*, pages 195–212, 2010.

[Int01] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409, 2001.

[IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

[Jac01] P. Jaccard. Etude comparative de la distribution florale dans une portion des Alpes et du Jura, 1901.

[JJK+13] Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. Outsourced symmetric private information retrieval. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 875–888. ACM, 2013.

[JKS08] S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *S&P*, 2008.

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In JuanA. Garay and Roberto Prisco, editors, *Security and Cryptography for Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 418–435. Springer Berlin Heidelberg, 2010.

[Kai08] J. Kaiser. A plan to capture human diversity in 1000 genomes. *Science*, 319, 2008.

[KE09] William J. Krouse and Bart Elias. Terrorist watchlist checks and air passenger prescreening. In *Congressional Research Service*, 2009.

[KJLM08] M. Kantarcioglu, Wei Jiang, Ying Liu, and B. Malin. A Cryptographic Approach to Securely Share and Query Genomic Sequences. *IEEE Transactions on Information Technology in Biomedicine*, 12(5):606–617, 2008.

[KM10a] J. Katz and J. Malka. Secure text processing with applications to private dna matching. In *CCS*, 2010.

[KM10b] Jonathan Katz and Lior Malka. Secure text processing with applications to private dna matching. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 485–492, New York, NY, USA, 2010. ACM.

[KMR11] S. Kamara, P Mohassel, and M. Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011.

[KO12] Kaoru Kurosawa and Yasuhiro Ohtaki. UC-secure searchable symmetric encryption. In Angelos D. Keromytis, editor, *FC 2012: 16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 285–298, Kralendijk, Bonaire, February 27 – March 2, 2012. Springer, Berlin, Germany.

[KP13] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In Ahmad-Reza Sadeghi, editor, *FC 2013: 17th International Conference on Financial Cryptography and Data Security*, volume 7859 of *Lecture Notes in Computer Science*, pages 258–274, Okinawa, Japan, April 1–5, 2013. Springer, Berlin, Germany.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12: 19th Conference on Computer and Communications Security*, pages 965–976, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[KR87] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31:249–260, March 1987.

[Lan89] E. Lander. DNA fingerprinting on trial. *Nature*, 339(6225), 1989.

[Lin13] LinkedIn Help Center. Account Restricted. Webpage, Mar. 2013. `https://help.linkedin.com/app/answers/detail/a_id/1386`.

[LK08] M Lepinski and S Kent. Additional diffie-hellman groups for use with IETF standards, January 2008. RFC 5114.

[LNO12] Yehuda Lindell, Kobbi Nissim, and Claudio Orlandi. Hiding the input-size in secure two-party computation. *IACR Cryptology ePrint Archive*, 2012:679, 2012.

[Mal05] B. Malin. An evaluation of the current state of genomic data privacy protection technology and a roadmap for the future. *Journal of the American Medical Informatics Association*, 12(1), 2005.

[MBC13] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. PIRMAP: Efficient Private information Retrieval for MapReduce. In *FC*, 2013.

[MLB12] B. Mood, L Letaw, and K. Butler. Memory-Efficient Garbled Circuit Generation for Mobile Devices. In *FC*, 2012. `http://fc12.ifca.ai/pre-proceedings/paper_71.pdf`.

[MPP10] Mark Manulis, Benny Pinkas, and Bertram Poettering. Privacy-preserving group discovery with linear complexity. In *Proceedings of the 8th international conference on Applied cryptography and network security*, ACNS'10, pages 420–437, Berlin, Heidelberg, 2010. Springer-Verlag.

[MS00] B. Malin and L. Sweeney. Determining the identifiability of DNA database entries. In *AMIA*, 2000.

[MS01] B. Malin and L. Sweeney. Re-identification of DNA through an automated linkage process. In *AMIA*, 2001.

[Nat11] National Center for Biotechnology Information (US). Restriction Fragment Length Polymorphism (RFLP). http://1.usa.gov/pha5sw, 2011.

[Nat16] National Center for Biotechnology Information (US). Single Nucleotide Polymorphism Database. http://www.ncbi.nlm.nih.gov/projects/SNP/, 2016.

[NDCD+13] Marcin Nagy, Emiliano De Cristofaro, Alexandra Dmitrienko, N. Asokan, and Ahmad-Reza Sadeghi. Do i know you?: Efficient and privacy-preserving common friend-finder protocols and applications. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, pages 159–168, New York, NY, USA, 2013. ACM.

[NHG12] NHGRI. DNA Sequencing Costs – Data from the NHGRI Large-Scale Genome Sequencing Program. http://www.genome.gov/sequencingcosts, 2012.

[Nie97] Jakob Nielsen. *Usability Engineering*. 1997.

[NPG14] Muhammad Naveed, Manoj Prabhakaran, and Carl A Gunter. Dynamic searchable encryption via blind storage. In *35th IEEE Symposium on Security and Privacy, 2014*, pages 639–654. IEEE Computer Society Press, 2014.

[Ope16] OpenSSL. http://www.openssl.org/, 2016.

[OU98] Tatsuaki Okamoto and Shigenori Uchiyama. A new public-key cryptosystem as secure as factoring. In *EUROCRYPT*, 1998.

[P+11] V. Pappas et al. Private search in the real world. In *ACSAC*, 2011.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[PB08] A. Prat and J. Baselga. The role of hormonal therapy in the management of hormonal-receptor-positive breast cancer with co-expression of her2. *Nature Clinical Practice Oncology*, 5(9), 2008.

[Pol13] Andrew Pollack. Justices Consider Whether Patents on Genes Are Valid. http://nyti.ms/XB7Tf9, 2013.

[PP12] G. Putzer and Y. Park. Are Physicians Likely to Adopt Emerging Mobile Technologies? Attitudes and Innovation Factors Affecting Smartphone Use in the Southeastern United States. *HIM*, 2012.

[Pre12] Presidential Commission for the Study of Bioethical Issues. PRIVACY and PROGRESS in Whole Genome Sequencing. http://www.bioethics.gov/cms/sites/default/files/PrivacyProgress508.pdf, 2012.

[PRZB11] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)*. ACM, October 2011.

[PVK+14] Vasilis Pappas, B Vo, F Krell, SG Choi, V Kolesnikov, A Keromytis, and T Malkin. Blind Seer: A scalable private DBMS. In *35th IEEE Symposium on Security and Privacy, 2014*, pages 359–374. IEEE Computer Society Press, 2014.

[R. 16] R. Roberts. REBASE, The Restriction Enzyme Database. `ftp://ftp.neb.com/pub/rebase/commdata.txt`, 2016.

[RJ10] C. Rotimi and B. Jorde. Ancestry and disease in the age of genomic medicine. *The New England journal of medicine*, 363(16), October 2010.

[RVBM09] Mariana Raykova, Binh Vo, Steven M Bellovin, and Tal Malkin. Secure anonymous database search. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 115–126. ACM, 2009.

[S+09] P. Stenson et al. The human gene mutation database: 2008 update. *Genome Medicine*, 1(1), 2009.

[SBC+07] Elaine Shi, John Bethencourt, T-HH Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 350–364. IEEE, 2007.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[Sha07] Hovav Shacham. A cramer-shoup encryption scheme from the linear assumption and from progressively weaker linear variants. Cryptology ePrint Archive, Report 2007/074, 2007. `http://eprint.iacr.org/`.

[Sin12] E. Singer. Democratizing DNA Sequencing. `http://www.technologyreview.com/biomedicine/26850`, 2012.

[Siv08] N. Siva. 1000 Genomes project. *Nature biotechnology*, 26(3), 2008.

[SK10] J. Sarasohn-Kahn. *How smartphones are changing health care for consumers and providers*. California HealthCare Foundation, 2010.

[SOJH09] S. Sankararaman, G. Obozinski, M. Jordan, and E. Halperin. Genomic privacy and limits of individual detection in a pool. *Nature Genetics*, 41(9), 2009.

[SSS12] Emil Stefanov, Elaine Shi, and Dawn Song. Policy-enhanced private set intersection: Sharing information while enforcing privacy policies. In *Public Key Cryptography–PKC 2012*, pages 413–430. Springer, 2012.

[SW81] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147, 1981.

[SWH12] E. Schadt, S Woo, and K. Hao. Bayesian method to predict individual SNP genotypes from gene expression data. *Nature Genetics*, 2012.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55, Oakland, California, USA, May 2000. IEEE Computer Society Press.

[TBHB11] H.K. Tabor, B.E. Berkman, S.C. Hull, and M.J. Bamshad. Genomics really gets personal: How exome and whole genome sequencing challenge the ethical framework of human genetics research. *American Journal of Medical Genetics*, 2011.

[The11] The Federal Bureau of Investigation. Combined DNA Index System (CODIS). `http://www.fbi.gov/about-us/lab/codis`, 2011.

[TMvR86] Andrew S. Tanenbaum, Sape J. Mullender, and Robbert van Renesse. Using sparse capabilities in a distributed operating system. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 558–563, 1986.

[TPKC07] J. Troncoso-Pastoriza, S. Katzenbeisser, and M. Celik. Privacy preserving error resilient dna searching through oblivious automata. In *CCS*, 2007.

[Tsa06] Tsung-Hsi Tsai. Average case analysis of the boyer-moore algorithm. *Random Struct. Algorithms*, 28:481–498, July 2006.

[UWL+09] Osman Ugus, Dirk Westhoff, Ralf Laue, Abdulhadi Shoufan, and Sorin A Huss. Optimized implementation of elliptic curve based additive homomorphic encryption for wireless sensor networks. *arXiv preprint 0903.3900*, 2009.

[vABKW08] Marco von Arb, Matthias Bader, Michael Kuhn, and Roger Wattenhofer. VENETA: Serverless friend-of-friend detection in mobile social networking. In *IEEE Conference on Wireless & Mobile Computing, Networking & Communication*, 2008.

[Ver11] Damien Vergnaud. Efficient and secure generalized pattern matching via fast fourier transform. In Abderrahmane Nitaj and David Pointcheval, editors, *Progress in Cryptology AFRICACRYPT 2011*, volume 6737 of *Lecture Notes in Computer Science*, pages 41–58. Springer Berlin / Heidelberg, 2011.

[vLSD+10] P. van Liesdonk, S. Sedhi, J. Doumen, P. H. Hartel, and W. Jonker. Computationally efficient searchable symmetric encryption. In *Proc. Workshop on Secure Data Management (SDM)*, pages 87–100, 2010.

[VPH+15] Mayank Varia, Benjamin Price, Nicholas Hwang, Ariel Hamlin, Jonathan Herzog, Jill Poland, Michael Reschly, Sophia Yakoubov, and Robert K. Cunningham. Automated assesment of secure search systems. *Operating Systems Review*, 49(1):22–30, 2015.

[W+09] R. Wang et al. Learning your identity and disease from research papers: information leaks in Genome Wide Association Study. In *CCS*, 2009.

[WH04] A. Weston and L. Hood. Systems biology, proteomics, and the future of health care: toward predictive, preventative, and personalized medicine. *Journal of Proteome Research*, 3(2), 2004.

[Wol13] Richard Wolf. Justices rule human genes cannot be patented. `http://www.usatoday.com/story/news/nation/2013/06/13/supreme-court-gene-breast-ovarian-cancer-patent/2382053/`, 2013.

[WWL+09] R. Wang, X. Wang, Z. Li, H. Tang, M. Reiter, and Z. Dong. Privacy-preserving genomic computation through program specialization. In *CCS*, 2009.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164. IEEE Computer Society, 1982.

[You12] S. Young. Knome Software Makes Sense of the Genome. `http://www.technologyreview.com/news/428179/knome-software-makes-sense-of-the-genome/`, 2012.

[ZPL+11] X. Zhou, B. Peng, Y. Li, Y. Chen, H. Tang, and X.F. Wang. To Release Or Not To Release: Evaluating Information Leaks in Aggregate Human-Genome Data. In *ESORICS*, 2011.