# PUTTING SECURE COMPUTATION TO WORK

by

**JASON M. PERRY**

A dissertation submitted to the
Graduate School–New Brunswick
Rutgers, the State University of New Jersey
In partial fulfillment of the requirements
For the degree of
Doctor of Philosophy
Graduate Program in Computer Science
Written under the direction of
Professor Rebecca N. Wright
And approved by

New Brunswick, New Jersey
October, 2015

ABSTRACT OF THE DISSERTATION

# Putting Secure Computation to Work

By JASON M. PERRY

Dissertation Director:
Professor Rebecca N. Wright

The demand for solutions that enable secure computation in distributed systems is only increasing. However, the current state of secure computation "in the wild" is highly unsatisfactory: provably secure solutions receive little attention, while untested security technologies with questionable security claims are being broadly marketed and deployed.

The classical approach to secure computation is secure multi-party computation (MPC) protocols, which allow a set of parties to jointly compute any given function while provably preserving the privacy of inputs and correctness of the output. Due to their generality, MPC protocols subsume a wide range of secure computation scenarios. However, MPC's adoption rate in the real world is extremely low. One reason is that MPC protocols have complex and differing security definitions; another is that MPC protocols typically do not map cleanly onto existing network application architectures.

Another promising research direction is secure protocols tailored for specific types of computations, such as text search and database access control. Such protocols attempt to strike a balance between efficiency and security, often by allowing a quantified amount of leakage. However, the practical security level of many of these protocols is not well understood.

Our research aims to make secure computation more deployable and trustworthy by bridging the gap between theoretical and applied secure computation. This dissertation presents 1) a systematization of MPC protocols that helps clarify their security and efficiency properties, 2) new, efficient protocols for access control in private databases, and 3) new security analyses, including statistical attacks, for currently used searchable encryption technologies.

# Acknowledgments

Thanks to Professor Rebecca Wright for her patient support and wise guidance for the past four years. It has been a pleasure working with you.

Thanks to Prof. David Cash for being a great "second advisor" and for encouragement and friendship. Thanks to committee members Dov Gordon and Prof. Vinod Ganapathy for helpful feedback.

Thanks to my collaborators, Giovanni di Crescenzo, Debayan Gupta, Joan Feigenbaum, Ritu Chadha, Jason Chiang, Allen McIntosh, Debbie Cook, and Thimios Panagos. Working together with you was easy and joyful. Special thanks go to Konstantine Arkoudas for true intellectual fellowship and the best internship I ever had.

Thanks to Prof. Chung-Chieh (Ken) Shan for invaluable mentoring and for showing me that I was already a researcher. Thanks to Prof. Matthew Stone for sound advice on moving forward after I was "orphaned", and to Prof. William Steiger and Carol DiFrancesco for helping steer me through the complications of the PhD process. Thanks to Prof. William Pottenger for support. Thanks to Prof. Jurek Jaromczyk, who encouraged me to aim high.

Shout-outs to my good friends at Rutgers, Drs. and soon-to-be-Drs. Rajat Roy, Mark Dilsizian, Sasho Nikolov, Chathra Hendahewa, Fatma Durak, Abdul Basit, Darakhshan Mir, and Edinah Gnang. Your encouragement was always a lifeline, whether you knew it or not.

Special thanks to Prof. Henry Kim for the inspiration that set everything in motion and made this day possible. Thanks to John and Grace Park, Moses and Mary Kim, and Samuel and Aromy Kim for their sacrificial support.

Thanks to my constant friends John Park, David Gates, Jun Hee Lee, Peter Kim, Joel Park, Luke Hong, Joseph Lee, Challenge Kim, and Shawn Ahmad-Chaustre.

Most special thanks to Mom, Dad, Patrick, and Mamaw for the environment of unconditional love that made everything possible.

Biggest thanks go to my wife Anastasia for believing in me and sticking with me through the years of uncertainty and slow progress, and for lunchboxes beyond number. I love you. Thanks to our kids Peter, Hope, and Andrew for bearing with my late nights and frequent absences.

Lastly, but firstly, I would like to acknowledge my Lord and Savior, Jesus Christ. Mark 13:31.

## Acknowledgment of Previous Publications

The work presented in Chapters 3 and 4 of this dissertation has appeared in partial form in [PGFW14b] and [DCFG$^+$14], respectively. The work in Chapter 5 is joint with David Cash, Paul Grubbs, and Thomas Ristenpart, and is currently in submission.

# Contents

# Chapter 1

# Introduction

*Who needs secure computation?*

The answer is: everybody needs secure computation! A secure computation protocol is called for whenever mutually distrustful (or less-than-completely-trustful) parties need to collaborate to compute some function of their joint data, without relying on a trusted party.

Just a few of the potential application scenarios for secure computation are: auctions and online deal-making, personalized medicine that preserves the privacy of patient data, governmental inter-agency data sharing, and encrypted-but-searchable cloud storage.

The goal of the work presented in this dissertation is to enable the effective deployment of secure computation in such real-world situations. The results presented address multiple factors, not limited to security and efficiency, but also deployment and decision-making factors.

## 1.1 State of the Field

As we are concerned with the adoption and deployment of secure computation in the real world, the first step is to examine the current state of the field in terms of both protocol research and deployments of secure protocols.

### 1.1.1 Specific or general protocols?

Many, if not most, of the problems in the domain of secure and private computing can be treated as an instance of *Secure Multi-Party Computation* (MPC). Any general MPC protocol can be seen as *emulating* a *trusted third party.* All of the applications of secure computation mentioned above are trivially solved if there is a third party that all of the other parties completely trust, with whom they each have a secure communication channel. Then all that needs be done is for each party to send their private input data to the trusted party, who computes the result and distributes the outputs to the parties, revealing nothing further. Insofar as any general MPC protocol emulates such a third party, it can be used as a solution for these problems.

The first general results showing that general MPC was possible were developed in the mid-80s. Since then, a steady stream of results has explored numerous aspects of the problem space, produced refined definitions and proofs, and achieved remarkable efficiency gains. Even since a paper titled "Secure Two-Party Computation is Practical" was published in 2009 [PSSW09], remarkable leaps in the efficiency of MPC have been made. However, we have also seen that "practical" in this sense does not immediately translate to "gainfully deployed".

In parallel, research has been carried out to propose secure protocols for specific types of functionality in specific configurations of machines and networks. This includes work in the areas of private databases and private information retrieval (PIR), oblivious RAM (ORAM), and searchable encryption. These technologies will be further surveyed in Chapter 2.

Thus, a central question for anyone needing to solve a secure computation problem is whether to apply a general MPC protocol or seek a solution tailored to the specific problem. Each approach has its own pluses and minuses.

Historically, the most obvious reason to choose a specific protocol over a general one is that the specific protocol may be more efficient. However, as mentioned, that gap is being narrowed by recent work in highly efficient general MPC protocols. Efficiency is no longer the only issue in play.

An advantage of the general MPC approach is in these protocols, security is inherent to the *protocol* rather than the *functionality.* This implies that as long as the general MPC protocol is correctly implemented and its proof of security takes environmental conditions into account, *any* computation undertaken using that protocol is secure. In contrast, developing a new application-specific secure computation protocol requires a new demonstration of its security. A disadvantage of the application-specific approach that is not inherent to the approach itself, but has a large practical impact, is that there may be a greater temptation to develop and deploy an application-specific protocol without giving sufficient time for a security proof and thorough security analysis. This is a reality that will come to light especially through the results presented in Chapter 5.

On the other hand, a significant potential advantage of specific protocols is that they may be easier to deploy. An application-specific protocol may be easier to fit to an existing network configuration; it may even be *legacy-compliant*, which means that it can be deployed by within a pre-existing service by its users without any changes required by the service provider. There are provably-secure protocols for specific computations that are easier to fit into existing client-server configurations, yet still have very broad applicability, such as search and retrieval. In general MPC, the system must seemingly be built from the ground up with the capability of executing MPC protocols in mind. This includes expressing the functionality in the model of computation required by the MPC protocol, which is typically a circuit.

### 1.1.2 Current State of Secure Computation Deployments

In reference to the positives and negatives of application-specific versus general secure computation solutions discussed above, the current state of deployment in some ways represents the worst of both worlds.

There are still very few deployments of provably secure general MPC protocols, and most of them have been one-time experiments. These include the well-known (within the field) Danish beet auction [BCD$^+$09] and the Taulbee survey experiment [FPRSJ04]. This lack of adoption is by no means due to lack of interest or effort on the part of MPC researchers. Efforts to deploy MPC more broadly continue to the present day, notably work involving the Sharemind system [Bog13, BKLPV13] and the recent commercialization of the SPDZ protocol and its descendants [DKL$^+$13].

Some of the reasons for low adoption of general MPC were discussed in the previous section. An additional barrier, perhaps the hardest to overcome, is that MPC protocols are themselves highly complex theoretical objects. It is difficult to precisely understand the security achievements of any given protocol and how it may apply in the real world. This is the primary driver for the work presented in Chapter 3.

In the domain of application-specific protocols, the problem seems like the "dual" of the problem with general MPC: there is broad marketing and deployment of products that do not provide the security they claim to. This is the case not merely because of security holes caused by software bugs; there are security products that implement fundamentally unsound protocols with no security proof at all.

A second cause of insufficient security in application-specific deployments is poorly understood *leakage* profiles of protocols that do have some proof of security. Some application-specific protocols have proven security that includes a quantified amount of leakage. However, though the leakage is quantified, the real-world ramifications of this leakage may not be well understood. One of the major contributions of the work in Chapter 5 is to demonstrate how exploitable such leakage is in real settings.

## 1.2 Approach and Outline of Results

To help remedy this regrettable situation, the work in this dissertation pushes on the problem from both sides: making protocols with full theoretical security more deployable, and making application-specific protocols more secure in spite of their leakage. We feel that when research progress goes in parallel on both the theoretical and applied fronts, they both benefit from the feedback and cross-pollination, and better bridges are built between cryptography and real-world security.

We begin with a survey of relevant research results in Chapter 2. In Chapter 3, we present a systematization of the space of general MPC protocols. In Chapter 4, we present highly efficient protocols for one particular application: enforcing access control in private databases. In Chapter 5, with the help of statistical attacks, we systematically investigate the security of protocols for secure search of outsourced client data. This includes solutions under the heading of *Symmetric Searchable Encryption*, as well as more ad-hoc protocols currently being deployed. With the perspectives given by these contributions, we conclude in Chapter 6 with observations and recommendations regarding the continued advancement of secure computation.

## 1.3 Preliminary Definitions and Notations

We close the introductory chapter by presenting terminology and notation that applies to all varieties of secure computation protocols and will be used throughout the dissertation. Further definitions relevant to specific results are presented in the appropriate chapters.

A function over the set of natural numbers is *negligible* if for all sufficiently large natural numbers $\sigma \in \mathcal{N}$, it is smaller than $1/p(\sigma)$, for any polynomial $p$.

A secure computation protocol is carried out by a set of *parties*, which are computational agents. The number of parties is usually denoted $n$. The parties are connected by a set of communication *channels*; usually a lossless channel is assumed to exist between every pair of parties. Whether the channel needs to be private—that is, immune to eavesdropping—depends on the protocol. The security of most protocols depends on the assumption that the parties are computationally bounded; the number of computational steps that any party can perform and the number of bits it can communicate is bounded by some polynomial in the input size.

The input of party $j$ to the secure computation is denoted $x_j$. The goal of a secure computation is to compute a functionality $f$ of those inputs, with outputs being returned to the parties themselves. We write a functionality, which may be probabilistic, as $f(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$. A secure computation protocol will typically be denoted by $\pi$. We abuse notation somewhat by using $\pi$ for both protocols for computing a specific functionality $f$, and for protocols that allow any functionality of a given form to be computed.

The security of a protocol $\pi$ is defined with respect to an *adversary* ($A$) who successfully *corrupts* some subset of the parties, fully controlling those parties and with access to all their protocol transcripts. Parties that are not corrupted are referred to as *honest*.

The set of subsets of corrupted parties that a protocol can tolerate while remaining secure is referred to as its *adversary structure*. Since the majority of protocols presented in the literature tolerate any subset of corrupted parties up to a given fraction of $n$, we generally refer to the corruption *threshold*, understanding that the definitions can also apply to general adversary structures.

We consider two fundamental security requirements for a secure computation, *privacy* and *correctness*.

- **Privacy**: A protocol $\pi$ is private if for every honest party $i$, no subset of corrupted parties smaller than the threshold can gain additional information on $x_i$ as a result of participating in $\pi$, apart from what can be inferred from the output.
- **Correctness**: A protocol $\pi$ is correct if no subset of corrupted parties smaller than the threshold can cause any honest party to receive an incorrect output of the functionality.

Note that the above definitions say nothing about what an adversary can accomplish by its choice of inputs for the corrupted parties; there is nothing we can do about that. A protocol is said to have *computational security* if either its privacy or correctness depends on computational assumptions and limitations; otherwise, its security is called *unconditional*.

The security of protocols is also defined with respect to the allowed actions of the adversary. The two most common adversary types are *honest-but-curious* and *malicious*. An honest-but-curious adversary is one that follows $\pi$, providing correct outputs at each step, but may try to gain additional information from the corrupted parties' transcripts. A malicious adversary is one that may take arbitrary actions (within its computational limits) to break the protocol's security.

Further definitions will be introduced as they are needed. Of course, the ideal is that a protocol provably meets these security requirements. This is not always achieved, and sometimes *relaxations* of the full security definitions are purposefully introduced for purposes of efficiency.

# Chapter 2

# Survey of Related Work

As the literature that applies to secure computation is so extensive, this survey is of necessity highly selective. The largest body of related work described here consists of results in the area of general multiparty computation, which we abbreviate as MPC.

We begin with an overview of preliminary cryptographic results on which many general MPC and application-specific secure computation protocols are built. Then we highlight some of the most significant papers of the general MPC literature. Next we survey papers that focus on implementation issues for MPC protocols and experiments in real-world deployment. Lastly, we give a selection of work describing application-specific secure computation protocols that relate most directly to the work in this dissertation.

## 2.1 Underlying Cryptographic Constructions

In this section we cover a subset of the seminal results in cryptography that comprise the backbone of many secure computation protocols, as well as higher-level constructions that have points of intersection with MPC results.

### 2.1.1 Pseudorandom Functions and Permutations

Two of the most fundamental types of cryptographic functions used for data hiding and authentication are *Pseudorandom Functions* (PRFs) and *Pseudorandom Permu-*

*tations* (PRPs). These are used as “black boxes” in many higher-level cryptographic protocols. Keyed PRPs and PRFs are currently implemented by efficient block ciphers and hash functions, though there is currently no proof that they satisfy the definitions of PRPs and PRFs.

For a detailed presentation of these concepts, refer to Katz & Lindell [KL14].

### 2.1.2 Secret Sharing

The technique of secret sharing using polynomial interpolation that underlies many MPC protocols was introduced by Shamir [Sha79]. In Shamir’s scheme, a polynomial of degree $t$ is used to share an element of the polynomial’s underlying field among $n$ parties, such that no subset of less than $t + 1$ parties can gain any information about the secret. Thus Shamir secret sharing achieves perfect (information-theoretic) privacy.

Many additional secret sharing schemes have been proposed and applied to MPC. An important advance for MPC was the introduction of “Verifiable Secret Sharing” by Chor et al. [CGMA85]. VSS protocols allow secret-shared values to be reconstructed even if some players are *malicious*, that is, that may arbitrarily deviate from the sharing protocol. This is key for constructing MPC protocols with security against maliciously corrupted parties, Cramer et al. [CDM00] showed how general MPC can be constructed from any linear secret-sharing scheme, a category into which many known secret-sharing schemes fall. Later, Cramer et al. [CDI05] showed how to convert shares between different secret-sharing schemes, providing efficiency gains for MPC.

### 2.1.3 Byzantine Agreement

Protocols for Byzantine Agreement enable a group of communicating parties, some of which may be dishonest, to agree on a value, such that all honest parties are guaranteed to hold the true value. An overview of the problem with algorithms is given in [LSP82]. Unfortunately, perfectly secure Byzantine agreement cannot be achieved in the case that $n/3$ or more of the parties are corrupted. Optimal algorithms achieving this bound were given in [FM88].

In the context of MPC protocols, Byzantine agreement can be used as a substitute for a *broadcast channel*, which allows a party to broadcast a single value to all other parties, with the parties being assured that the same value has been received by all of them.

### 2.1.4 Oblivious Transfer

Oblivious Transfer (OT) is a specific type of multi-party computation in which a sender holding multiple inputs communicates one or a subset of those inputs to a receiver, in such a way that the receiver cannot learn which inputs were received.

The original definition is due to Rabin [Rab81]. The generality of the OT operation was demonstrated by Killian [Kil88], who showed that general MPC can be unconditionally constructed on the assumption of Oblivious Transfer.

OT protocols with full security against malicious adversaries seem to be expensive to achieve. The first protocol for adaptively secure Oblivious Transfer was presented by Beaver [Bea98]. The age of modern efficient OT protocols, requiring only a constant number of group exponentiations per OT, started with Naor and Pinkas [NP01].

The major breakthrough in making OT usable in practical protocols was *OT Extension*, also presented first by Beaver [Bea96]. This allows a large number of oblivious transfers to be efficiently computed from just a small number of "seed" OTs, using only the assumption of one-way functions.

A framework for oblivious transfer allowing multiple levels of security in semi-honest and malicious models, in both UC and non-UC versions, appears in [PVW08]. The current most efficient single OT protocol is given by Chou and Orlandi in [CO15].

An interesting variation on OT is the protocol presented by Rivest [Riv99], which makes use of a *semi-trusted* third party to accomplish a highly efficient Oblivious Transfer. The third party does not see any of the sender's input data nor find out which data the receiver obtained. This result is relevant to the work in Chapter 4 on distributing trust in private database systems.

### 2.1.5 Homomorphic Encryption

Homomorphic encryption schemes allow certain types of computations to be carried out on encrypted data by parties that cannot efficiently decrypt it. The recent discovery by Gentry [Gen09] of the existence (given suitable assumptions) of fully homomorphic encryption holds promise for many secure computing applications. Using FHE directly to achieve MPC is still far more expensive than other means, but insights related to homomorphic encryption have driven the development of more efficent MPC protocols, e.g. SPDZ [DPSZ12].

## 2.2 General Multi-Party Computation Protocols

In this section we highlight some of the key achievements in general MPC protocols. The presentation is chronological and focuses first on the introduction of new conceptions of security, before jumping to describe some of the currently most efficient protocols. The systematization given in Chapter 3 provides a framework for thoroughly understanding and comparing these and many more results in general MPC.

### 2.2.1 Major Achievements in MPC

The first protocols for secure multi-party computation were presented by Yao [Yao82, Yao86]. Though the papers do not present any general MPC protocols, Yao is credited with the idea of combining multi-party protocols for addition and multiplication to form *garbled circuits* by which any function can be computed securely.

The seminal result showing the possibility of computationally-secure multiparty computation among three or more parties is by Goldreich et al. [GMW87]. This work utilizes the verifiable secret sharing protocol of [CGMA85] and describes a "protocol compiler" for transforming protocols secure against semi-honest adversaries into protocols secure against malicious adversaries.

Unconditionally-secure MPC was introduced in [BGW88, CCD88]. These results show that as long as fewer than $n/3$ parties are corrupted (even maliciously) and a secure channel exists between every pair of parties, MPC with perfect security can be achieved. Definitions of statistically secure MPC, which is less than perfect security yet does not require computational assumptions, were given in Rabin and Ben-Or [RB89].

A significant step in provable security for MPC was definitions and proofs of *Universally Composable* (UC) security, given by Canetti [Can00, Can01]. A UC-secure MPC protocol can be considered secure regardless of the environment in which it executes, and whether or not it is composed concurrently or sequentially with other MPC protocol executions. The first constructions of universally composable MPC protocols, requiring a *Common Reference String* or CRS, were presented in [CLOS02].

An intriguing related problem, Secure Multiparty Computation of Approximations, was treated by Feigenbaum et al. [FIM+01]. This can be seen as a generalization of 'exact' MPC. Proving that an approximating MPC is secure has unique difficulties, because an approximation can reveal *more* data about the parties' inputs than the true answer.

### 2.2.2 Modern protocols

The current state-of-the-art protocols combine ideas from earlier information-theoretic and assumption-based protocols. The most efficient protocols are given in what is known as the *preprocessing* model, which means that the most expensive part of the computation can be carried out by the parties in advance, before either the function to be computed or its inputs is specified. This allows protocols that are secure against a malicious adversary with very little additional *online* cost over security against an honest-but-curious adversary.

The beginning of this era of MPC may be considered to begin with Damgard et al. [DO10], followed and improved upon by Bendlin et al. [BDOZ11]. Significant efficiency improvements in the offline phase were achieved in [DPSZ12]. This was the first protocol in which each multiplication gate requires a number of operations in the offline phase that is less than quadratic in the number of parties. Refined versions of this protocol, known as "SPDZ", are currently being implemented and commercialized.

## 2.3 MPC Implementations and Deployment Experiments

The work presented in this dissertation shares common goals with research meant to foster the real-world adoption of secure multi-party computation. Thus, works describing MPC implementations and in-the-field experiments are of particular significance.

The first known implementation of secure two-party computation is Fairplay [MNPS04], which also provided a compiler from a C-like language to Boolean circuits This achievement generated much interest but was not widely adopted, being seen largely as a proof-of-concept. It was followed in 2008 by FairplayMP [BDNP08], the first known implementation of multi-party computation based on the constant-round protocol of [BMR90]. A somewhat successful field experiment was carried out by Feigenbaum et al in 2004 [FPRSJ04].

A novel approach to implementing two-party computation was put forth in the TASTY framework [HKS+10], which uses a programmable hybrid of garbled circuits and homomorphic encryption, and allows users to program protocols directly, not merely functionalities. The next step was VIFF [DGKN09], which was an attempt to overcome one efficiency bottleneck by minimizing the number of synchronization points required for MPC.

The current most flexible, actively developed, publicly available framework for secure multiparty computation is Sharemind, described most thoroughly in Bogdanov's PhD thesis [Bog13]. It began as an implementation of three-party semi-honest secure computation in Bogdanov et al. [BLW08] and grew from there. More recent experiments are described in Kamm et al. [BKLPV13].

The SPDZ protocol of Damgard et al. [DPSZ12] has also been implemented, with experiments and improvements to the original protocol described in [DKL+13]. Its source code is not publicly available, however.

Information from this section was partially derived from a report by Segal [Seg13].

## 2.4 Secure Data Access Protocols

The work described here is for application-specific protocols, yet of a kind having very broad use cases, as they provide functionality for secure data access.

### 2.4.1 Private Information Retrieval

Private information retrieval, or PIR, is a secure computation in which a client retrieves records from a database held by a server, without the server learning which records were retrieved. In the case that the client may only access a single record from the database, it is equivalent to Oblivious Transfer. The trivial algorithm is for the client to download the entire database, but clearly this has a communication complexity equal to the size of the database. The first demonstration that PIR could be achieved with less communication, by using multiple servers, is by Chor et al. [CKGS98]. The first protocol for single-database, computationally private information retrieval was by Kushilevitz and Ostrovsky [KO97].

### 2.4.2 Private Databases

On the more applied side, numerous efforts have sought to add efficient security and privacy features to the classical relational database model, without achieving the strict guarantees of PIR. MIT's CryptDB [PRZB11] integrates a number of security and privacy techniques into an SQL database platform. The IARPA SPAR project produced significant new research in privacy-preserving databases, including the "Blind Seer" system by Pappas et al [PKV$^{+}$14] and the SPADE private database developed by ACS [DCCMP14], with which the work in Chapter 4 is integrated.

### 2.4.3 Oblivious RAM

Oblivious RAM (ORAM) protocols combine private information *retrieval* with private information *storage*, hiding both the reads and writes of a client from a server holding a set of data. The pioneering work is by Goldreich and Ostrovsky [GO96].

Both PIR and ORAM are more efficient for their specific tasks than a general MPC solution would be; in particular, the communication complexity of non-trivial PIR and ORAM is sub-linear in the size of the database. MPC protocols typically assume that the entire inputs of all parties must be shared.

However, ORAM may still be too slow for applications with large-scale datasets. The best ORAM solutions have overhead that is logarithmic in the size $N$ of the database, while the only solution that would be considered truly scalable for large datasets would have overhead that does not depend on $N$ at all. This is a primary motivator for the work presented in Chapter 5.

### 2.4.4 Searchable Encryption

We can say that "Searchable Encryption is to ORAM as Private Databases are to PIR." That is, research in searchable encryption seeks to solve the practical problem of search on outsourced data in a more efficient manner than ORAM, by allowing for a controlled amount of leakage.

The most well-known solutions of this type in the literature are referred to as Symmetric Searchable Encryption (SSE), as they make use of highly efficient symmetric encryption techniques, e.g., block ciphers. The field was pioneered by Song et al. [SWP00], who introduced the idea of "hiding until search". In this scenario, a client uploads her dataset in a form that is opaque to the server until such time that the client issues a search query. The goal of the SSE scheme is to minimize the amount of additional information revealed to the server by each search operation.

The modern era of index-based SSE constructions was ushered in by Curtmola et al. [CGKO06]. A fully-length-hiding SSE scheme by means of intertwined linked lists was presented by Cash et al. [CJJ$^+$13]. The authors also present new constructions for Boolean queries.

A structured classification of searchable encryption schemes that shows their strengths and vulnerabilities is presented in Chapter 5.

# Chapter 3

# A Systematization of General Multiparty Computation Protocols

A major barrier to the adoption of general MPC is the sheer number of results and protocols that have been presented in the field, each with different parameters and providing different types of security guarantees. The purpose of the research presented in this chapter is to provide a way to get a handle on this body of results, with the goal of making it more usable in real-world applications.

We propose a framework for organizing and classifying research results in the active field of secure multiparty computation (MPC). Our *systematization of secure computation* consists of (1) a set of definitions circumscribing the MPC protocols to be considered; (2) a set of quantitative axes for classifying and comparing MPC protocols; and (3) a knowledge base of propositions specifying the known relations between axis values. We have classified a large number of MPC protocols on these axes and developed an interactive tool for exploring the problem space of secure computation. We also give examples of how this systematization can be put to use to foster new research and the adoption of MPC for real-world problems.

## 3.1 Introduction

For more than 30 years, since the groundbreaking results of Yao [Yao82, Yao86] and Goldreich *et al.* [GMW87], hundreds of research papers on *Secure Multiparty*

*Computation* (MPC) have appeared, many of them proposing original protocols for carrying out general secure computation under varying sets of assumptions. In this chapter, we systematically organize the main research results in this area, in order to:

- Help potential users of MPC learn which existing protocols and implementations best match the sensitive-data computations they would like to perform. This may stimulate adoption of MPC in areas where it would be beneficial.
- Help new researchers get up to speed in a complex area by providing an overview of the "lay of the land."
- Help MPC researchers explore the problem space and discover remaining openings for protocols with new combinations of requirements and security features—or for new impossibility results that preclude the existence of such protocols.

Most research papers in MPC include comparisons of their results to related work, often with tables related to the most significant protocol features, in order to provide context for understanding the paper's contributions. However, none has attempted to organize the larger problem space in order to meet the goals listed above. There are a handful of introductory surveys and textbook-like treatments of MPC [Gol04, CD05, HL10, CDN13]; these have (justifiably) focused on a narrower region of the problem space or specific security model, in order to present the material in a pedagogically clean way. In contrast, we do not limit ourselves to one model or set of definitions but instead provide a framework for examining their variations.

In Sections 3.1.1, 3.2, and 3.3, we present the three major components that we believe are needed to systematize the main body of MPC results: (1) a set of definitions delineating the boundaries of the problem space; (2) a set of quantitative features for describing protocols; and (3) a knowledge base of propositions specifying the known relationships and dependencies among features. In Section 3.4, we describe the construction of a systematization database of more than 60 significant MPC protocols. In Section 3.5 we present a user interface designed to aid the exploration of the database of systematized MPC protocols and show how our systematization can be put to work to facilitate new research. Finally, in section 3.6 we describe how these efforts have thus far been transitioned to a community-maintained model.

### 3.1.1 Definitions

As a necessary prerequisite for this work, we have carried out an extensive literature survey producing an annotated bibliography of MPC research. It contains over 180 papers from the MPC literature, as well as a sampling of important papers for related problems, including secret sharing and oblivious transfer.

In addition to a written paragraph annotating each paper, the BibTeX source of our Annotated Bibliography includes tags for each paper indicating which aspects of MPC it treats, *e.g.*, `2party` for a paper with a specifically two-party protocol, or `uncond` for a protocol with unconditional security. These tags make it possible to write scripts to automatically generate a bibliography for any specific sub-problem or aspect of MPC.

The bibliography continues to be updated on an ongoing basis. The most recent version is available online [PGFW14a].

### 3.1.2 Definition of a general MPC protocol

In this section, we provide definitions to delimit the scope of the secure computation protocols that we are concerned with. These definitions are purposely quite broad, so that a large range of work can potentially be captured by them.

Variables that determine the fundamental nature of an MPC protocol include: (1) whether the protocol is for a fixed number of parties (most commonly, two) or is for any number $n \geq 2$ of parties, (2) whether the protocol is for computing one specific functionality (*e.g.*, set intersection) or any of a class of functionalities, and (3) whether the protocol treats exact computation only or secure computation of approximations, which is a generalization of exact MPC [FIM+01]. Our initial literature survey encompassed all of these, although not exhaustively. For the current systematization, we consider only protocols for exact computation, and thus we have four definitions, one for each setting of the two variables.

Since the way that security is defined varies from protocol to protocol, and indeed a primary purpose of our systematization is to examine such variations, our definitions necessarily cannot give any fixed definition of security. What matters is that, in the literature, protocols are proven to meet rigorous definitions of security and that our systematization indicates which definitions are actually in use for a given protocol. Therefore, for an MPC protocol to be considered as a candidate for systematization, in addition to fitting into one the definitions listed below, it must be accompanied by rigorous definitions of security, including privacy of inputs and correctness of outputs, that the protocol has been proven to meet. The nature of these security definitions is elaborated in the next subsection.

**Definition 1.** *A protocol for* Secure $n$-party Computation of a functionality $f$ *is a specification of an interactive process by which a fixed number $n$ of players, each holding a private input $x_i$, can compute a specific, possibly randomized, functionality of those inputs $f(x_1, ..., x_n) = (y_1, ..., y_n)$.*

**Definition 2.** *A protocol for* Secure Multiparty Computation of a functionality $f$ *is a specification of an interactive process by which any number $n \geq 2$ of players, each holding a private input $x_i$, can compute a specific, possibly randomized, functionality of those inputs $f(x_1, ..., x_n) = (y_1, ..., y_n)$.*

**Definition 3.** *A protocol for* Secure $n$-party Computation of a class $\mathcal{C}$ of functionalities *allows a fixed number $n$ of players, each holding a private input $x_i$, to compute an agreed-upon, possibly randomized, functionality of those inputs $f(x_1, ..., x_n) = (y_1, ..., y_n)$, where $f$ is any member of the class $\mathcal{C}$ of functionalities.*

**Definition 4.** *A protocol for* Secure Multiparty Computation of a class $\mathcal{C}$ of functionalities *allows any number $n \geq 2$ of players, each holding a private input $x_i$, to compute an agreed-upon, possibly randomized, functionality of those inputs $f(x_1, ..., x_n) = (y_1, ..., y_n)$, where $f$ is any member of the class $\mathcal{C}$ of functionalities.*

The class $\mathcal{C}$ is typically used to refer to the model of computation in which a protocol's functionalities are represented, such as circuits or RAM programs. A majority of the work in MPC has been concerned with universal (Turing-complete) computation, but there has been work exploring secure computation specifically for restricted computation classes, such as AC0 or NC1 circuits or regular or context-free languages.

All of the protocols we surveyed for the systematization fall under one of these definitions, with the majority coming under the most general definition (Definition 4). Yao-like two-party computation protocols fall under Definition 3.

## 3.2 Linear axis representation of protocol features

The main conceptual object in our systematization is a set of linear *axes*, where each axis represents an ordering of values of a single feature of MPC protocols. Every axis has at least two labeled values, at the endpoints. Some axes are continuous and others discrete. MPC protocols can be scored on these axes, allowing them to be compared quantitatively. This is the first attempt we are aware of to factor research results in MPC into such a representation. The axes were selected based on our literature survey, using two guiding principles:

1. The axes should be as orthogonal as possible, minimizing overlap (although some logical dependencies between axes are unavoidable.)
2. The set of axes should be *complete* in the sense that they can express all distinctions of security and (asymptotic) efficiency between any two protocols.

For any discrete axis that is not inherently binary, the number of occupied intermediate values on the axis is subject to change. The diagrams below show intermediate values that are known to have been achieved by MPC protocols. However, this should not be seen as finally determining the number of points on the axis. Indeed, one of the objectives of the axis representation is to highlight the possibility of future protocols with new intermediate values. This has already happened for several axes over the history of MPC research. For example, the appearance of protocols tailored for covert adversaries, such as those of Aumann *et al.* and Goyal *et al.* [AL07, GMS08], showed that there are intermediate values along the "Maliciousness" axis (Axis 7), whereas previously only the passive/malicious distinction had been considered.

We orient all the axes in the same direction, such that moving from left to right on a given axis indicates an improved protocol—*e.g.*, one that is more efficient, has a stronger security guarantee, or requires a weaker setup or computational-hardness assumption. In drawing the non-numeric axes, the points have been placed with equal spacing; the relative distances between points on these axes should not be considered significant.

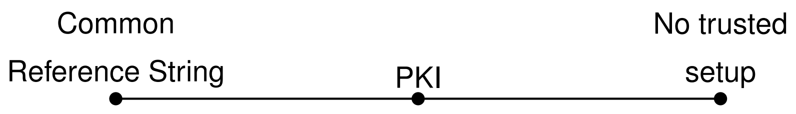
Our axes do not include the model of computation in which a protocol is expressed. This is a categorical feature which is indicated by the definition (Section 3.1.1) under which the protocol falls. The model of computation for each protocol is included in its entry in our MPC protocol database, described in Section 3.4. We have not generated axes for proof techniques, because a proof technique is not a function of the protocol; a protocol's security may be proven in a number of ways.

We now proceed to describe the axes and values in detail. The axes are informally grouped into four categories, which serves to highlight the tradeoffs inherent in achieving secure multiparty computation. The axes in categories I and II, Environmental features and Assumptions, can be thought of as what one "pays" to enable secure MPC, and categories III (Security) and IV (Efficiency) can be seen as what one is "buying." A similar tradeoff structure can also be seen on a smaller scale among axes within the efficiency category. When it is helpful, our description of an axis also cites particular MPC solutions that instantiate points on the axis.

I. Environmental Features Axes

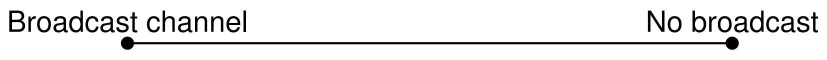
   This is the category of features assumed to be provided by the execution environment. The right endpoint of these axes indicates that the feature is not required in any form.

   1. Trusted setup

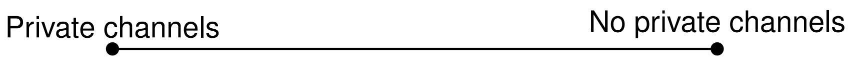
      Protocols achieving the highest composable security levels require some type of trusted data to be shared by all the parties prior to the protocol execution. The middle point, PKI, is occupied by protocols such as that of Barak *et al.* [BCNP04], who showed how to use public-key-like assumptions instead of a polynomial-length common reference string in any case where computational security suffices.

   2. Broadcast

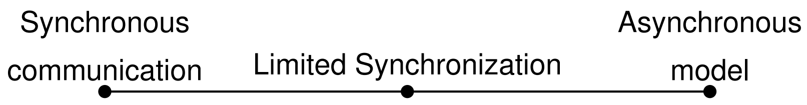
      The broadcast-channel assumption means that each party has the ability to send a message to all other parties simultaneously, and that all parties receiving the broadcast have assurance that the same message was received by all parties.

3. Private channels



The private channel assumption is only significant for unconditionally secure protocols, because cryptography using basic computational-hardness assumptions can be used to emulate private and authenticated communication channels.

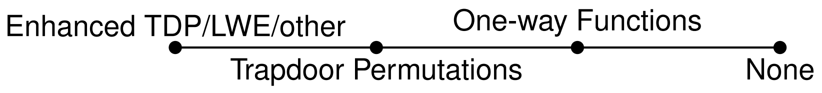
4. Synchronization



A basic assumption of the early MPC protocols is that they operate on a *synchronous* network, in which all sent messages arrive on time and in order. The asynchronous case was first considered in Ben-Or *et al.* [BCG93], where messages may be arbitrarily delayed and arrive in any order. Note that, in such a case, it is impossible to know whether a corrupted party has failed to send a message or, rather, the message is simply delayed. Later works, such as that of Damgård *et al.* [DGKN09], have staked out intermediate points on this axis by giving protocols that require a smaller number of synchronization points (typically a single one).

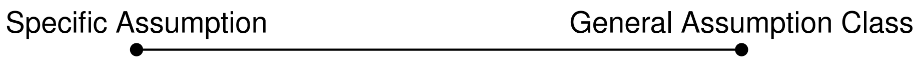
II. Assumption Axes

5. Assumption level



A total (linear) ordering for cryptographic assumptions is not known, and the separation of assumptions cannot currently be unconditionally proven. We therefore use broader categories of assumption type, because these are usually sufficient to distinguish protocols. If a protocol makes no such assumptions, it is said to have *unconditional* security (see Axis 10). Some work specifies protocols in a *hybrid* model, with no concrete computational assumptions, but in which some high-level cryptographic operation (such as oblivious transfer) is assumed to exist as a black box. See Section 3.4 for a discussion of how such protocols are treated in this systematization.

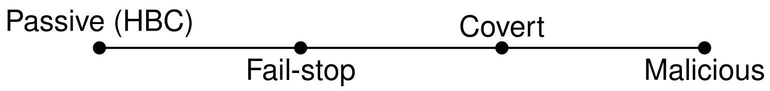
6. Specific or general assumption



Some more efficient protocols have been designed by making use of specific number-theoretic assumptions. This axis indicates whether the protocol requires such assumptions or whether it is stated so as to use any assumption from a given class, *e.g.*, trapdoor permutations.

III. Security Axes

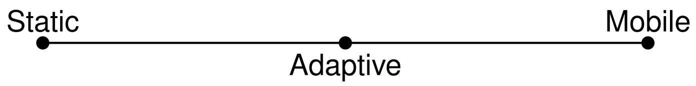
7. Adversary maliciousness



A passive, or honest-but-curious, adversary is one that follows the protocol but may use the data of corrupted parties to attempt to break the protocol's privacy. A fail-stop adversary follows the protocol except for the possibility of aborting. A malicious adversary is one whose behavior is arbitrary, and a covert adversary is like a malicious one, except that it only deviates from the protocol if the probability of being caught is low.

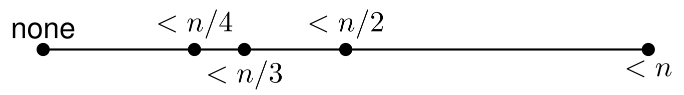
Not present on the axis is the value "rational," since the class of rational adversaries is in fact a generalization that can encompass the entire axis, except for fully malicious, because malicious behavior can be truly arbitrary. The position of a rational adversary on the axis is determined by its utility function.

8. Adversary mobility



A static adversary must choose which parties to corrupt before the protocol begins. An adaptive adversary can choose which parties to corrupt, up to the security threshold (see Axis 9), over the course of the computation, after observing the state of previously corrupted parties. A mobile adversary is able to move corruptions from one party to another in the course of the computation.

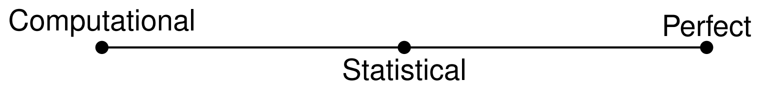
9. Number of corrupted parties tolerated



This is the maximum number of corrupted parties for which the (strongest) security guarantees of the protocol hold. The values shown are chosen merely to be representative of the most well known protocols; any value along the axis is possible.

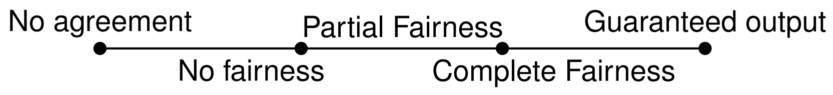
Some protocols tolerate additional corrupted parties at a lower level of maliciousness; see axes 13 and 14.

10. Security type



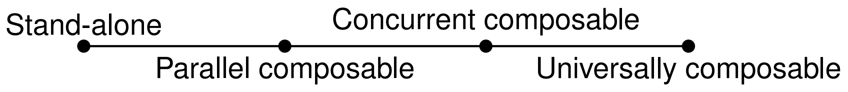
Both statistical and perfect security are unconditional, that is, not based on computational hardness assumptions. Note that true unconditional security typically cannot be achieved through the internet, even if an unconditionally-secure protocol is used, since all unconditionally secure protocols require the assumption of either private or broadcast channels, which on the internet must be emulated by cryptography.

11. Fairness guarantee



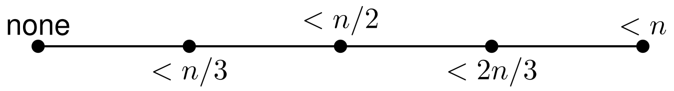
A protocol is *fair* if all honest parties receive the output if any party does. Agreement means that either all honest parties receive the output or none of them do. Protocols without agreement were introduced by Goldwasser *et al.* [GL02]. Some authors use the term "with abort" to refer to the no-fairness situation, in which dishonest parties can abort after receiving the correct output.

12. Composability



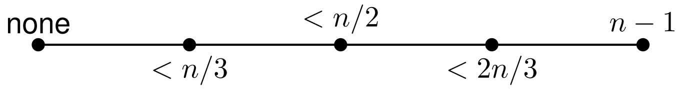
The composability guarantees of a protocol indicate whether that protocol remains secure when executed in an environment where other protocols may be executed sequentially or in parallel. The strongest guarantee, *universal composability* (UC), implies that the security properties of a protocol hold regardless of the environment in which it is executed.

13. Bound for additional passively corrupted parties tolerated



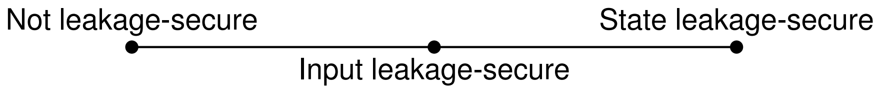
This axis applies to protocols achieving "mixed adversary" security. A protocol that tolerates a certain proportion of maliciously/covertly corrupted parties may also tolerate an additional number of passively corrupted parties, up to a certain threshold. The values on this axis represent that upper threshold. This and the following axis relate to MPC protocols with "graceful degradation", which is surveyed in Hirt *et al.* [HLMR11].

14. Corrupted parties tolerated with weakened security



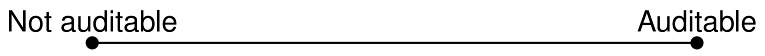
This axis applies to protocols with "hybrid security" results. A protocol that tolerates a certain proportion of corrupted parties (Axis 9) may in fact tolerate a larger number of corruptions, but with a weaker security type, *e.g.*, computational vs. unconditional security.

15. Leakage Security



Leakage security is an additional guarantee that an adversary cannot gain an advantage even if it can force all honest players to "leak" some bits of information about their state in the course of the computation. See definitions in Bitansky *et al.* [BCH12].

16. Auditability

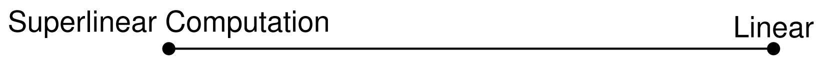


This axis indicates whether the protocol includes computations that allow for examining the transcript of computation after it is finished, to prove that the parties have correctly followed the protocol. This may be the most recent axis to come into existence, starting with a result of Baum *et al.* [BDO14]

IV. Efficiency Axes

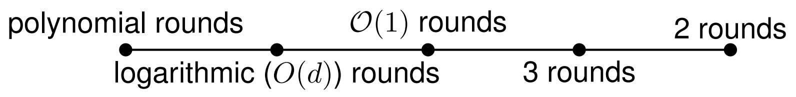
Our efficiency axes are concerned primarily with the asymptotic efficiency of the protocol in question.

17. Online computational overhead



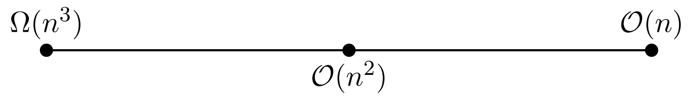
Historically, the main efficiency concern in MPC has been with communication rather than computational complexity; thus the lack of elaboration of this axis. More recently, Ishai *et al.* have notably shown how to achieve MPC with constant computational overhead [IKOS08], and the RAM-model results of Gordon *et. al* [GKK$^+$12] have shown the possibility, in the RAM model, of MPC with amortized computation that is sublinear in the input size.

18. Online communication complexity (rounds)



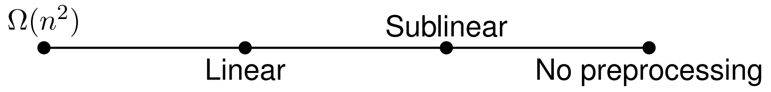
Here, $d$ is the depth of the circuit representing the functionality. Minimizing the number of rounds of computation, independently of the total amount of bytes communicated, is crucial for efficiency in a high-latency or asynchronous network environment. Fully general MPC (with three or more parties) was shown to require at least three rounds in Gennaro *et al.* [GIKR02], although for some functionalities a 2-round protocol is possible.

19. Online communication complexity (per-gate)



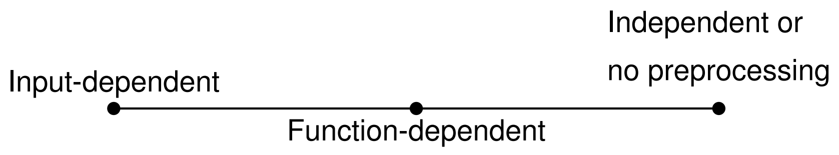
This is the most significant measure of efficiency for MPC protocols. It can represent either bits or field elements of communication. The original BGW protocol has a communication complexity of $O(n^6)$ bits per multiplication gate in the worst case. Anything cubic or worse occupies the lowest position on our axis, as finer distinctions at that level would have little value for distinguishing current, more practical protocols.

20. Preprocessing Communication complexity



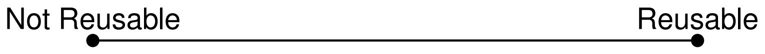
Many recent protocols achieve improved online communication efficiency by means of a preprocessing phase. In the case where the functionality is represented as an arithmetic circuit, the preprocessing phase is typically a simulation of a trusted dealer that distributes *multiplication triples*, which allow local evaluation of multiplication gates in the online phase. Sublinear preprocessing typically indicates that the preprocessing consists only of exchange of public keys, which is also indicated as a setup assumption (Axis 1).

21. Preprocessing Dependency



In some protocols, the preprocessing phase depends on the specific functionality to be computed, while in others it only depends on the upper bound of the size of the circuit. In all cases, the preprocessing is independent of the parties' inputs.

22. Preprocessing Reuse



This indicates whether the information computed in the preprocessing stage, of whatever type or amount, can be reused for multiple computations. Data of the nature of a public key typically can be reused, while *e.g.,* garbled circuits traditionally cannot be reused without breaking security. But see a recent result of Goldwasser *et al.* [GKP$^+$13].

As mentioned above, we have endeavored to make this selection of axes as complete as possible. The value of completeness in a systematization can be illustrated as follows: Suppose there are two MPC protocols whose scores along the axes are identical for every axis except one. If the set of axes is complete, then we can be confident that the protocol with the higher value on that axis is strictly better.

## 3.3 MPC theorems as axis dependencies

Since many MPC protocols involve essential tradeoffs in order to achieve security or efficiency, a systematization of secure computation also needs to model what is known about how features of protocols interact. In this section, we present the second major aspect of our systematization: a list of each of the theorems known to imply constraints among the axes' values, each accompanied by a statement of the constraint. References are given to the paper in which the theorem implying the constraint was proven.

**Theorem 1** ([BGW88]). *If statistical or perfect security is obtained, then either a broadcast channel or private channels must be assumed.* ***Axis constraint****: If Axis 10's value is to the right of "Computational," then either Axis 3's value is "Private channel" or Axis 2's value is "Broadcast channel".*

**Theorem 2** ([RB89]). *No protocol with security against malicious adversaries can tolerate more than $n/2$ corrupted parties without losing the complete fairness property.* ***Axis constraint****: If Axis 7's value is "Malicious" and Axis 9's value is to the right of $n/2$, then Axis 11's value must be to the left of "Complete fairness".*

**Theorem 3** ([Cle86])**.** *No protocol unconditionally secure against malicious adversaries can guarantee output delivery with $n/3$ or more corrupted parties. **Axis constraint**: If Axis 7's value is "Malicious," Axis 10's value is to the right of "Computational," and Axis 9's value is to the right of "n/3," then Axis 11's value must be to the left of "Guaranteed output".*

**Theorem 4** ([BGW88])**.** *No protocol can have perfect security against more than $n/3$ maliciously corrupted adversaries. **Axis constraint**: If Axis 7's value is "Malicious" and Axis 9's value is to the right of $n/3$, then Axis 10's value must be to the left of "Perfect".*

**Theorem 5** ([GIKR02])**.** *Any general MPC protocol (with three or more parties) with complete fairness against a malicious adversary must have at least three rounds. **Axis constraint**: If Axis 7's value is "Malicious" and Axis 11's value is at or to the right of "complete fairness", then Axis 18's value must be to the left of "2 rounds".*

**Theorem 6** ([BGW88])**.** *For unconditional security against $t$ maliciously corrupted players, $n/3 \leq t < n/2$, a broadcast channel is required. **Axis constraint**: If Axis 10's value is to the right of "Computational" and Axis 7's value is "Malicious" and Axis 9's value is to the right of $n/3$, then Axis 2's value must be "Broadcast channel".*

**Theorem 7** ([GMW87])**.** *For (even cryptographic) security against $\geq n/3$ maliciously corrupted players, either a trusted key setup or a broadcast channel is required. **Axis constraint**: If axis 7's value is "Malicious" and Axis 9's value is to the right of $n/3$, then either Axis 2's value must be "Broadcast channel," or else Axis 1's value is to the left of "No trusted setup."*

**Theorem 8** ([BGW88])**.** *There can be no unconditionally secure protocol against an adversary controlling a majority of parties. **Axis constraint**: Axis 10's value can be to the right of "Computational" only if Axis 9's value is at or to the left of $n/2$.*

**Theorem 9** ([CKL03])**.** *There is no protocol with UC security against a dishonest majority without setup assumptions. **Axis constraint**: If Axis 9's value is to the right of $n/2$ and Axis 12's value is "Universally composable," then axis 1's value must be to the left of "No trusted setup".*

**Theorem 10** ([BCG93])**.** *In an asynchronous environment, there is no protocol with guaranteed output secure against a fail-stop adversary corrupting $n/3$ or more parties.*

***Axis constraint****: If Axis 4's value is "Asynchronous," Axis 7's value is at or to the right of "Fail-stop," and Axis 11's value is at "Guaranteed output," then Axis 9's value must be at or to the left of* $n/3$*.*

**Theorem 11** ([BCG93])**.** *In an asynchronous environment, there is no protocol with guaranteed output secure against a malicious adversary corrupting* $n/4$ *or more parties.* ***Axis constraint****: If Axis 4's value is "Asynchronous," Axis 7's value is "Malicious," and Axis 11's value is at "Guaranteed output," then Axis 9's value must be at or to the left of* $n/4$*.*

One validation of our choice of axes is that these theorems are directly and compactly expressible in terms of them, thus giving a unified representation of the central body of knowledge of MPC. The axis constraints can easily be represented in a programming or knowledge representation language, as Section 3.5 shows.

## 3.4 An Extensible Protocol Database

We scored more than 60 of the most significant protocols in secure multiparty computation on our axes, integrating information from our annotated bibliography, resulting in an extensible *MPC protocol database*.

Many papers in the area include multiple protocols. We give each protocol a separate entry in the database, which is labeled by adding a suffix to the usual "alpha"-style reference. For instance, "[GMW87]-mal" refers to the protocol of [GMW87] that is secure against a malicious adversary. The database also indicates whether an implementation of the protocol is known to exist.

The work of constructing the database motivated many revisions of our set of axes and highlighted difficulties in systematizing MPC results, some of which we discuss here.

**Efficiency.** As mentioned in the axis descriptions, our efficiency axes are concerned primarily with asymptotic efficiency measurements. When we populated our database, we relied on evaluations in the literature, frequently from the paper actually introducing the protocol.

The model of computation in which a protocol's functionalities are expressed can have a large impact on concrete efficiency. Historically, MPC functionalities have been expressed as circuits. The original Yao model considers Boolean circuits, while most of the current state-of-the-art MPC protocols are in the arithmetic-circuit model. Implementing these requires performing field arithmetic, and, although the size of the field elements is a constant in the security parameter, the time taken to perform field operations can have a significant impact on efficiency. Although this difference in concrete efficiency is not captured by the axes, our protocol database notes the model of computation for each protocol.

Even in the asymptotic case, comparing the efficiency of MPC protocols is an extremely difficult problem because of multiple interacting aspects of efficiency present in MPC. In selecting an actual implementation, a concrete analysis and/or empirical efficiency measurements should also be consulted.

**Substitutability.** One factor that makes it nontrivial to enumerate a list of MPC protocols is that many protocols described in the literature make use of subprotocols for cryptographic operations in a black-box fashion, making it possible to substitute different protocols implementing that operation. This can alter not only the performance characteristics but also the computational and environmental assumptions and security and composability guarantees of the resultant protocol. In some cases, a new and improved subprotocol can trivially be used to improve an older MPC protocol, but no published work explicitly presents the improvement; in other cases a protocol explicitly allows for black-box substitution of subprotocols, in which case it is said to be stated in a *hybrid* model. In the case of the *OT-hybrid* model, in which oblivious transfer is a black box, recent work in *OT extension* has produced significant performance gains.

An extreme case of this "substitutability" factor is the IPS compiler of Ishai *et al.* [IPS08], which is not only in the OT-hybrid model but also allows any of a wide of of honest-majority MPC protocols to be plugged in as an "outer protocol," with the resulting protocol inheriting some (but not all) of the security properties of the outer protocol.

To address this complex issue, we have limited our systematization to represent only concrete instantiations of hybrid protocols. The axes are such that a protocol in a hybrid model must first be "instantiated" with concrete sub-protocols in order to be scored. In our database, we have sought to enumerate as many such concrete protocols as possible that are based on well-known hybrid-model protocols.

**Two-party secure computation.** Although much research has been done specifically addressing two-party secure computation, starting from Yao's original garbled circuits idea, it can be considered as a special case of multiparty computation, in which, if security against a malicious adversary is sought, no honest majority can be assumed (Axis 9 at $n - 1$). Thus, two-party protocols can at least theoretically be compared with multiparty results in this category.

In reality, however, vast improvements in efficiency have been made for the two-party case. We note that these optimizations often come at the expense of *symmetry*: The security guarantee against cheating by one of the two parties may be weaker than for the other. For instance, one of the two parties may be able to cheat with an inverse-polynomial probability, while the other may only be able to cheat with negligible probability. Asymmetry is not included in our axis definitions, and so two-party protocols are scored by the weaker of the two sets of security guarantees. Of course, the database indicates which protocols and implementations are strictly for two parties.

## 3.5 Interacting With the Protocol Database

As mentioned in Section 3.1, the main theorems of secure multiparty computation demonstrate essential tradeoffs involved in securely computing a functionality among distrustful parties. To leverage the information that our systematization captures about these tradeoffs and gain insight into the problem space, we first experimented with visually plotting the protocol axis scores of the MPC database, as described in Section 3.5.1. However, we found that the highly categorical nature of the data makes it difficult to gain insight from a static visualization. In the subsequent subsection, we describe an an interactive tool we have developed for interacting with the systematization and MPC protocol database, which provides a better way of coming to grips with the multi-dimensional landscape of MPC protocols.

### 3.5.1 Visualizing the space of MPC protocols

To understand the space of MPC protocols, we desire an informative view of the "lay of the land," and a deeper understanding of the tradeoffs required for secure computation. This can be done by visualization of the axis-scored protocols. In this case, understanding the relations between MPC protocols becomes a task of *Visual Data Mining* [Ins97], and choosing a protocol is a kind of multi-dimensional optimization.

Finding suitable visualizations of the axes data is non-trivial the data is highly categorical. Many features are binary, and most axes have no more than four or five points. One of the more appropriate visualization types is the spider, or radar, plot. Spider plots have non-orthogonal axes, to allow more axes to be compared than visual dimensions are available. Still, the number of axes in our systematization is too many for a readable spider plot; so we have chosen subsets that allow inspection of the significant tradeoffs between MPC protocols. See figure 3.1.

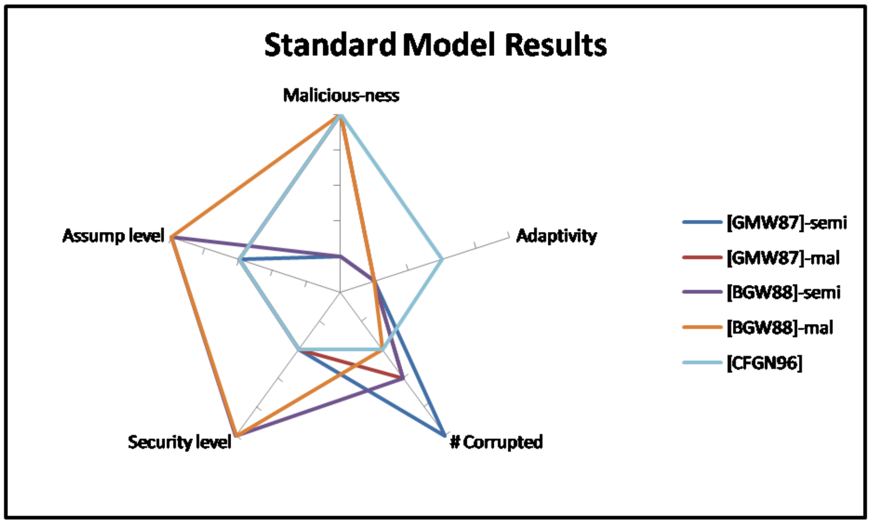


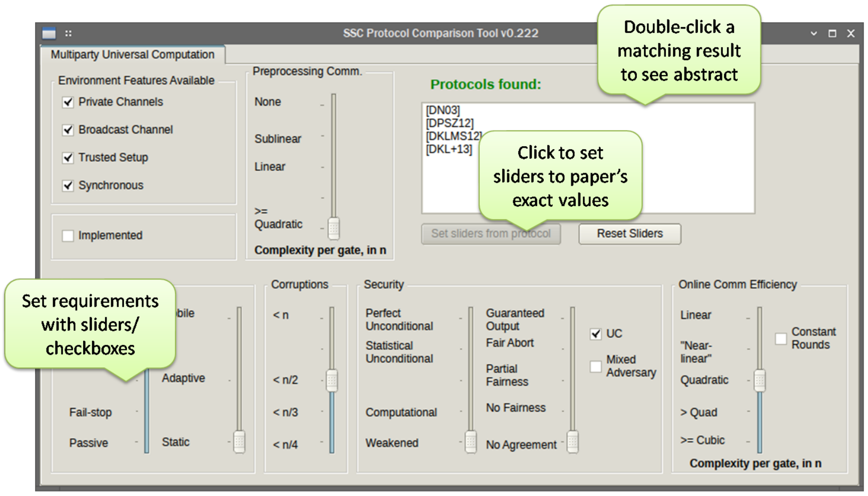
Figure 3.1: Sample spider plot comparing five MPC protocols in the standard model.

The value of these is limited because the scale is arbitrary, and the choice of a subset of axes must exclude some important considerations.

Other potentially useful visualizations include parallel coordinate plots, and back-to-back bar graphs (with axes from the 'pay' group on the left and from the 'buy' group on the right.)

### 3.5.2 A prototype decision-making support tool

We have developed a prototype GUI tool, *SysSC-UI*, which reads in a protocol database of axis values and enables the user to adjust a set of sliders and check-boxes corresponding to our axes of systematization. For the tool's interface, the axes are oriented vertically rather than horizontally as printed above, so a higher position of a slider corresponds to a stronger result. A dynamically updated *results window* displays the protocols from the database that match the specified axis values. See the screenshot in Figure 3.2. The source code for the desktop version is available online `https://code.google.com/p/syssc-ui/`, as well as a beta web-based version `http://work.debayangupta.com/ssc/`.



☑ Private Channels
☑ Broadcast Channel
☑ Trusted Setup
☑ Synchronous
☐ Implemented
☑ UC
☐ Mixed Adversary
☐ Constant Rounds

Protocols found:
[DN03]
[DPSZ12]
[DKLMS12]
[DKL+13]

Figure 3.2: Screenshot of the SysSC-UI tool for interacting with the protocol database.

For a given setting of the sliders and checkboxes, the results window shows all papers whose axis values are at the same level *or higher* than the settings. Thus, the tool presents all protocols that are *at least as good* as the specified settings. When the tool is started, the sliders and checkboxes are all set to the least constraining position, such that every protocol in the database is displayed in the results window. There is also a button to reset the tool to this state. Another button sets the sliders to the exact values of the protocol in the currently highlighted protocol, allowing the precise achievements of a protocol to be examined. This has the side effect of changing the output to the results window, so that only protocols that are at least as good as the selected one are displayed.

Double-clicking on a reference in the results window will display a pop-up window giving the authors' full names and the description of the paper containing the protocol from the annotated bibliography. The GUI also indicates when the positioning of the sliders is such that a secure computation protocol is known to be impossible, by means of an encoding of the theorems in Section 3.3.

The axes and values displayed by SysSC-UI are a subset of those in the full systematization. This was done in order to simplify the interaction and avoid confusion from the visual display of too much information at once. For example, for the composability axis, there is only a single checkbox, to indicate whether the protocol is proven universally composable or not.

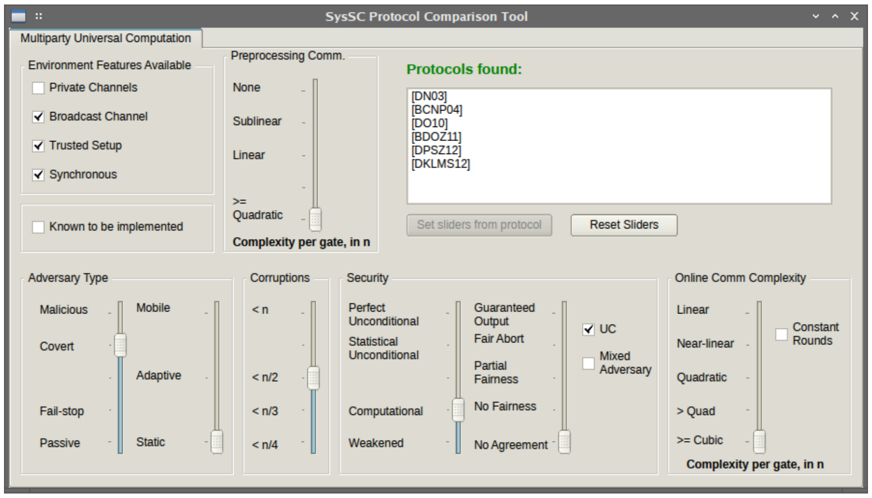
We now present sample use cases highlighting the features of SysSC-UI.

### 3.5.3 Sample use cases for SysSC-UI

**Finding the best protocol for a known problem.** Consider a scenario in which a technology consultant is hired by a company to find a way to compute some function of a distributed set of sensitive data residing on servers owned by different divisions of the company. We show how she can use the SysSC-UI tool to find an appropriate MPC protocol for achieving this secure computation.

In the initial state of the UI tool, all four environmental assumption boxes are checked, and all sliders are in the lowest position, so that every protocol in the database is displayed in the results window. To begin, our consultant unchecks "Private Channels", knowing that the computation will be carried out over an ordinary internet connection, which should always be assumed to be tapped. She wishes to protect against adversaries that are covertly malicious, so she moves the leftmost Adversary Type slider up to "Covert". The consultant is suspicious of protocols that use a weaker model to prove security, so she moves the first Security slider up to "Computational." Furthermore, she suspects that universal composability is necessary to guarantee security in a heterogenous environment, so she checks the "UC" box. The protocols resulting from these selections can be seen in the results window in Figure 3.3.

The consultant wishes to determine the most efficient protocol that meets these requirements, so next she moves the "Online Comm Complexity" slider up to the highest level for which the results box is not empty. We would like to achieve this online complexity with the minimum preprocessing complexity, so she tries sliding the "Pre-



☐ Private Channels
☑ Broadcast Channel
☑ Trusted Setup
☑ Synchronous
☐ Known to be implemented
☑ UC
☐ Mixed Adversary
☐ Constant Rounds

**Protocols found:**

[DN03]
[BCNP04]
[DO10]
[BDOZ11]
[DPSZ12]
[DKLMS12]

Figure 3.3: Results shown by the SysSC-UI tool after selections have been made.

processing comm" slider up. If it is moved up too far, the results window becomes empty. So she readjusts both this and the Online slider to find an agreeable trade-off between online and preprocessing complexity. The results of this exploration are shown in Figure 3.3.

**Exposing directions for cryptographic research.** In using the tool, one invariably stumbles upon a setting of the sliders / boxes that is not in the "known impossible" range, and yet has no papers with a matching protocol listed. Of course, one reason for this could simply be the incompleteness of the protocol database. Another possible reason is that the combination of features is not desirable from a security or efficiency standpoint. However, a third possibility exists, which is that a genuine opening for new research has been revealed. Two kinds of such "holes" may reveal new research directions: (1) Gaps between achieved and proven impossible security levels, and (2) settings in which a weakening of security parameters may allow greater efficiency. An example of the second type of advance is the case of positing composable security definitions weaker than universal composability, as in [PS04].

## 3.6 Ongoing Work

As it is unlikely that we will permanently be able to stay abreast of the growing MPC literature, we have developed a web-based survey that allows researchers to submit descriptions of new protocols and their features on the axes so that they can be integrated into the protocol database and SysSC-UI. The survey is available at `http://goo.gl/T4ORzr`. Community participation is vital to the continuing usefulness of this effort.

The set of theorems should also be expanded beyond those listed in Section 3.3 as knowledge of secure computation increases. This work has highlighted several remaining unknowns in characterizing the possibility of secure computation; for example, perhaps some of the malicious impossibility results hold for the covert case as well. A longer-term goal is to characterize the efficiency of the protocols more precisely, in terms of the number of elementary operations, to make the efficiency of all protocols directly comparable.

Modeling the flexibility of protocol frameworks such as IPS [IPS08] and in general protocols specified in some hybrid model that allow black-box substitution of subprotocols, would require a systematization at a higher level of abstraction.

## 3.7 Conclusion

Systematizations of knowledge are especially needed in research fields where a large body of results have been generated in a short time, and secure multiparty computation is undoubtedly such a field. Without an effort to systematically organize results, there may be unnecessary duplication of research efforts, the barriers to entry for new researchers may be needlessly high, and results may not see useful applications as early as they could. Our systematization of secure computation is a tool that can significantly ease the task of coming to grips with this sprawling body of results, and potentially speed its adoption in fields where it would be useful.

# Chapter 4

# Three-Party Protocols for Policy Enforcement in Private Databases

In this chapter we consider one particular class of multi-party computations: The operation of a distributed database with specific security and privacy requirements. In fact, the entire computation process of a secure database could be encapsulated as general secure computation and implemented using one or more of the protocols presented in the previous chapter. However, that would be too inefficient, especially for large databases used in the real world.

One specific problem in secure databases is that of checking a client's queries against a server's query *compliance policy*, denying results for non-compliant queries. The challenge is to find protocols that accomplish this without breaking the security and privacy requirements of the database. We call these protocols *Database Policy Compliance* (DPC) protocols.

We introduce DPC protocols for a specific three-party model, consisting of a client interested in making database queries, a server providing its database for client access, and a third party (e.g., a cloud server) holding the (encrypted) outsourced data and helping both other parties. The protocols are designed to compose with an underlying private database retrieval protocol (with no query compliance policy) in the same model.

We give formulations of the desired requirements, such as preservation of query correctness, compliance completeness, compliance soundness, privacy and efficiency.

Then we present new protocols that satisfy a natural combination of correctness, privacy, and efficiency requirements. The efficiency of these protocols has been demonstrated in a prototype implementation. Technical contributions of independent interest include the use of equality-preserving encryption to produce highly-practical symmetric-cryptography protocols (i.e., *one order of magnitude faster* than "Yao-like" protocols), and the use of a query rewriting technique that maintains the privacy of the compliance result.

### Outline of Results

In Section 4.1 we give an overview of the scenario and describe the features of data retrieval and database policy compliance protocols. In Section 4.2 we detail the definitions and models in which our results will be stated, including a formal definition for private database policy compliance protocols. In Section 4.3 we present our basic solution for keyword search query types in the semi-honest model and a proof that it meets our formal definition. In section 4.4, we briefly discuss extensions of the protocols for types of range queries, and a subset of possible range and Boolean policies.

In Section 4.5 we discuss the implementation of our basic solution, show measurements of the its running time during the query phase, and a derived analysis of the running time of our extension protocols. In Section 4.6, we draw conclusions that relate these solutions to the broader concerns of the dissertation.

## 4.1 Overview of the Three-party Model and Protocol Requirements

### 4.1.1 Secure and private databases

Any distributed database consists of at least two categories of communicating parties. *Servers* are parties holding a set of records, and *clients* are parties that issue queries to servers and receive matching records in response. A *secure and private database* is a distributed database that provides specific security guarantees to all parties. There is no one fixed set of such guarantees. Typically, privacy guarantees provide assurance that no other parties, including server parties, may gain information about clients' queries or the records accessed through those queries. This is generally accomplished by means of query and database encryptions having certain properties, as has been studied in the area of Private Information Retrieval.

Database applications with such privacy requirements will often also have security requirements for protecting sensitive records in the database against access by unauthorized clients. These requirements can be specified in the form of *access control policies*. Thus we have the additional need to enforce policies based on the content of specific queries, still without revealing the query to the server.

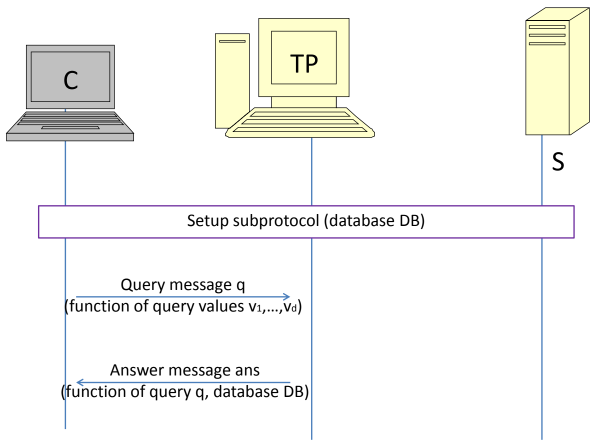
Our goal is to augment known encrypted database retrieval solutions with security properties, in particular policy-based query authorization, while preserving the privacy and efficiency properties of the basic database retrieval solution.

### 4.1.2 The three-party model

To achieve practically efficient solutions, we consider a 3-party model, including a client $C$ (interested in private data retrieval), a server $S$ (offering data for retrieval conditioned to a query compliance policy), and a third party $TP$ (e.g., a cloud server) helping both parties to achieve their goals. Since quite a few studies already address the problem of privacy-preserving data retrieval in a three-party model [CDdV$^+$05, HILM02, YZW06], we focus on the policy compliance building block alone. Moreover, we design protocols that are modularly compatible with a natural class of data retrieval protocols in the literature. We first assume database retrieval (DR) protocols in the 3-party model with a relatively general structure, as shown in Figure 4.1, and then investigate the modeling and design of database policy compliance (DPC) protocols that combine with known DR protocols, as shown in Figure 4.2.

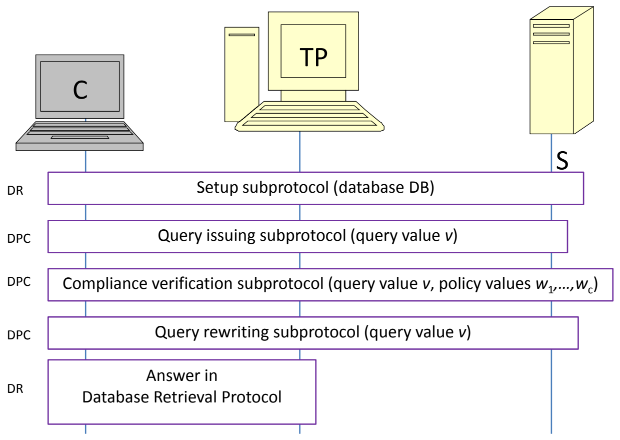
### 4.1.3 Database Policy Compliance protocol requirements

We use the following carefully formulated, but informal, requirements for DPC protocols:



C
TP
S
Setup subprotocol (database DB)
Query message q
(function of query values v1,…,vd)
Answer message ans
(function of query q, database DB)

Figure 4.1: Structure of a Database Retrieval Protocol



C
TP
S
DR
Setup subprotocol (database DB)
DPC
Query issuing subprotocol (query value v)
DPC
Compliance verification subprotocol (query value v, policy values w1,…,wc)
DPC
Query rewriting subprotocol (query value v)
DR
Answer in
Database Retrieval Protocol

Figure 4.2: Composition of a database-retrieval protocol with a database-policy-compliance protocol

1. *Preservation of Query Correctness*: A client that could retrieve all of the records that satisfy its query using a DR protocol can still do so if the query is compliant.

2. *Compliance Completeness*: All queries that satisfy (resp., do not satisfy) the policy are found to be compliant with probability 1 (resp., with negligible probability).

3. *Compliance Soundness*: For any efficient (and even malicious) adversary impersonating the client, the server can correctly compute (except with negligible probability) the policy compliance of whichever query message is received and answered by the third party according to the DR protocol.

4. *Privacy*: Privacy of database values, policy values and query values are preserved, in that information leaked to any honest-but-curious party may not exceed the following: the system parameters (which are intended to be known by all parties), the compliance bit $b$ intentionally revealed to $S$, the encrypted query message intentionally revealed to the third party, representing the true query in the case of a compliant query, or a random query matching no records in the case of a non-compliant query. As will be described, this is to reduce leakage of the policy-compliance result to $TP$ or $C$).

5. *Efficiency*: The protocol should have low time, communication and round complexity. One of the most significant design criteria we target to reduce computational overhead of query compliance checking is to minimize or eliminate costly public-key cryptographic operations and to achieve protocols faster than a mere application of secure function evaluation techniques.

Note that in addition to privately checking the compliance of a given query, we also wish to keep the policy hidden from the clients. Of course, it is impossible to prevent the client from gaining any information at all about the policy while still fulfilling queries, because the way a particular query is fulfilled necessarily reveals information about the policy. Still, in order to hide some additional information about the policy, we require that the result of a non-compliant query is indistinguishable from a query that matches zero records in the database, so that the protocol does not reveal to clients whether a query that returns no matches does so because it is non-compliant or because there are actually no matching records. Of course, sometimes a client will be able to distinguish these two cases due to auxiliary information.

### 4.1.4 Formulation of DPC as a Multi-Party Computation

The query-answering function of a distributed database can be viewed as a *reactive* multi-party computation between a client $C$ and server $S$. A reactive multi-party computation takes place in multiple stages, in which each stage is a separate computation, but the inputs of one or more parties may depend on the outputs of the previous stage. Further definitions for reactive MPC can be found in Goldreich [Gol04].

The functionality computed at stage $i$ is written $f_i$. Recall from Section 1.3 that $x_j$ is the input of party $j$ and $y_j$ is the output. The database is denoted $D$ and the query at stage $i$ (which may depend on the output of previous stages) is written as $q_i$. We describe stage $i$ of the database functionality as follows:

$$f_i(x_s = D,\ x_{C,i} = q_i) \rightarrow (y_{S,i} = \emptyset,\ y_{C,i} = D[q_i])$$

Note that the server party $S$ receives no output.

To formalize the policy compliance requirements, we expand the above functionality so that $S$ has an additional input, a policy $p$, and learns a compliance bit $b$, based on which he may return an empty set instead of the matching records.

$$f_i(x_S = (D, p),\ x_{C,i} = q_i) \rightarrow (y_{S,i} = b,\ y_{C,i} = D[q_i]||\emptyset)$$

This represents an idealized version of the functionality, with no leakage.

### 4.1.5 Policy Enforcement Protocols

We design three protocols for enforcing compliance of *keyword search* queries. We only consider *whitelist* (resp., *blacklist*) policy types, where the query is compliant only if the query value is equal to one (resp., none) of the policy values, with extensions to combinations via Boolean formulae. For such query and policy types, we provide highly efficient and scalable database policy compliance protocols which meet all our requirements, as detailed in Figure 4.3. An important property not captured in Figure 4.3 is that all our 3 DPC protocols only require $O(1)$ cryptographic operations per query and policy value, and are thus one order of magnitude faster than 2-party arbitrary function evaluation protocols [Yao86], as these require at least $\Omega(\ell)$ cryptographic operations per query and policy value, if $\ell$ denotes the length of these values (even in recent optimized solutions). Just like achieved previously for DR protocols in the 3-party model, our DPC protocols not only minimize or eliminate costly public-key cryptography operations, but they provide concrete time efficiency, which we document through performance numbers collected during our implementation efforts (in Section 4.5).

Our solutions rest on two main technical contributions: using equality-preserving

| Requirement | Protocol $\pi_1$ | Protocol $\pi_2$ | Protocol $\pi_3$ |
|---|---|---|---|
| **Correctness preservation** | if DR protocol satisfies Added Property 1 | if DR protocol satisfies Added Property 2 | if DR protocol satisfies Added Property 1 |
| **Compliance completeness** | unconditionally | unconditionally | unconditionally |
| **Compliance soundness** | (not satisfied) | unconditionally | unconditionally |
| **Privacy** | under PRP assumption (leaks repeated queries to S) | under PRP assumption (leaks repeated queries to S) | under PRP assumption |
| **Time and communication** | linear in policy size | linear in policy size | linear in policy size |
| **Round complexity** | $O(1)$ in policy size | $O(1)$ in policy size | $O(1)$ in policy size |

Figure 4.3: Security properties satisfied by the DPC protocols. Entries in the first four rows give the condition under which the property is satisfied. The PRP assumption is the existence of pseudo-random permutations (instantiated in a practical implementation as block ciphers.) Added Properties 1 and 2 of DR protocols are defined in Section 4.2.

symmetric encryption, secure under multiple encryptions[MH81], to reduce the number of cryptographic operations, and performing query rewriting whenever the query is not compliant in order to make the results of non-compliant queries indistinguishable from queries matching no records.

To the best of our knowledge, there is no previous work on privacy-preserving, efficient, *query* policy compliance checking for database queries. That is, although there has been previous work on 3-party protocols in which the data set being searched is encrypted, the query is kept private, and queries are only allowed if they satisfy certain conditions, we are unaware of previous, formal work in which the restriction on allowable queries (i.e., the policy) is supposed to be kept private. The majority of existing work in this area focuses on techniques that allow execution of queries on outsourced, encrypted data in which allowable queries are defined by conditions on the database attributes [HILM02, YZW06, CDdV$^+$05, EG07]. Most of these approaches use indexing information stored with the encrypted data to return the correct data in response to a query without decrypting the data itself. Ceselli *et al.* [CDdV$^+$05] also consider the trade-off between efficiency and vulnerability to linking attacks. A number of schemes [GSW04, LYCL11, dVFJ$^+$07] look at different kinds of access control in such systems.

Hamlen et al [HKK12] have independently described principles of secure policy enforcement for cloud data. Some of techniques they describe, such as query rewriting, are also expressed in our system, though this work does not focus on query compliance policies specifically.

All these areas mainly focus on privacy of the retrieved information which is different from the client authorization problem, on which we focus here.

## 4.2 Definitions for Privacy-preserving DR and DPC Protocols

In this section, we formalize the definitions and models that we use in our investigation of privacy-preserving policy-compliance protocols, and describe the security and efficiency properties that our protocols must satisfy. We begin by defining the relevant notions of data, query, policy, participant, network and protocol models, and completeness, soundness, privacy, and efficiency.

### 4.2.1 Preliminaries

**Data, Query and Policy Models.** We model a *database table* (briefly, database) as a matrix DB with $n$ rows and $m$ columns, where each row is associated with a data *record*, each column is associated with a data *attribute*, and each database entry $\mathrm{DB}(i, j)$ is the value of the $j$-th attribute of the $i$-th record. The database *schema* consists of $n$, $m$, and the *domains* of each of the $m$ attributes (*i.e.*, the $j$-th domain is the set of values that the $j$-th attribute of a record can take on), and is assumed to be known by all parties that participate in the protocol. We assume that domains are large in that a randomly chosen domain element is, with very high probability, not in DB. (If DB does not satisfy these conditions, then simple padding of domain strings can be used to make it so.)

A *query* $q$ contains a database attribute and a corresponding *query value* from the relevant attribute domain. Using SQL notation and terminology, and using $v$ to denote a query value, we initially consider keyword-match queries, *i.e.*, queries of the form

A server's *query compliance policy* (briefly, *policy*) contains, for each attribute $j$, $1 \leq j \cdots m$, a set $W_j = \{w_{j,1}, \ldots, w_{j,c_j}\}$ of *policy values* drawn from the $j$-th domain. All of the clients that access DB through this server are subject to the same policy. On input a query value $v$, an attribute name (or, equivalently, an attribute index $j$), and a set of attribute values $W_j$, the policy returns 1 (resp., 0) to denote query compliance (resp., non-compliance). We mainly consider the *whitelist* and *blacklist* policies:

1. *Whitelist*: If query $q$ refers to the $j$-th attribute, then $p$ returns 1 iff $v \in W_j$;
2. *Blacklist*: If query $q$ refers to the $j$-th attribute, then $p$ returns 1 iff $v \notin W_j$.

Intuitively, a blacklist policy captures the notion of a set of forbidden query values, while a whitelist policy restricts queries to a specified set of allowed values. We assume that the lengths $c_j$ of whitelists and blacklists and the lengths of the policy values $w_{j,k}$ are system parameters known to all parties.

**Participant, Network and Protocol Model.** We consider the following participant types, all assumed to run in (probabilistic) time polynomial in a common security parameter, denoted in unary by $1^\sigma$. The *client* is the party, denoted by $C$, that is interested in retrieving data from the database. The *server* is the party, denoted by $S$, that stores the database and the policy and is interested in allowing clients to retrieve data, whenever allowed by the policy. The *third party*, denoted as $TP$, helps the client to retrieve data and the server to enforce the query compliance policy. Each client is assumed to be capable of communicating with both the server and the third party. For simplicity, we assume a confidential and authenticated network (as achieved in practice by a network-security protocol such as TLS) with no packet loss.

This model encompasses *database-retrieval* (DR) protocols and *database policy-compliance* (DPC) protocols. This work builds upon DR protocols of a certain general form (exemplified by protocols proposed in, *e.g.*, [HILM02, YZW06]) that allow clients to retrieve data using a server and a third party when no query policy is in place; Figure 4.1 depicts the general structure that we require of a DR protocol. Our contribution in this chapter is the design and analysis of DPC protocols that can be modularly combined with DR protocols of this form in order to realize both the database-retrieval and the policy-compliance goals; see Figure 4.2.

### 4.2.2 Requirements of DR Protocols

As explained above, we require DR protocols to enable $C$ to retrieve records in a model in which it can communicate with both $S$ and $TP$ and in which there is no query policy. We further require a DR protocol to have three properties: a specific protocol structure, a query-correctness property, and a unique-query property. The DR protocol structure is depicted in Figure 4.1; its crucial features are as follows:

1. $C$, $S$ and $TP$ run a preliminary setup subprotocol

   (this enables $TP$ to later answer $C$'s query on the database owned by $S$)

2. Given a query $q$, $C$ constructs a *query message* Q and sends it to $TP$

3. $TP$ computes an *answer message* $\textsf{ans}$ and sends it to $C$

4. Based on Q and $\textsf{ans}$, $C$ can compute database records that satisfy $q$, if any.

The *unique-query* property requires that, for any database DB and any properly formatted query message Q, there is at most one pair $(\text{attribute\_name}, v)$ for which $C$ could have generated query message Q. When such a pair exists, we refer to $v$ as the "query value associated with Q." Note that, although there is at most one pair associated with each properly formatted Q, there are, in general, many Q's that $C$ might generate from a given pair, because $C$ is probabilistic.

The *query-correctness* property requires that, for any database DB, any input pair (attribute_name, $v$), and any Q with associated query value $v$, at the end of the DR protocol, $C$ can compute all records in DB that satisfy query attribute_name $= v$.

In integrating secure policy compliance, we will impose one of the following additional properties on DR protocols:

**Added DR Property 1.** At the end of step 1, $TP$ stores $F(k_{\text{c,s}}; DB(i,j))$, for each database entry $DB(i,j)$, where $F$ is a pseudo-random permutation and $k_{\text{c,s}}$ denotes a key shared between $C$ and $S$

**Added DR Property 2.** At the end of step 1, $TP$ stores the triple encryption $F(k_{\text{c,s}}; F(k_{\text{c,tp}}; F(k_{\text{c,s}}; DB(i,j))))$, for each database entry $DB(i,j)$, where $F$ is a pseudo-random permutation and $k_{\text{c,s}}$ (resp., $k_{\text{c,tp}}$) denotes a key shared between $C$ and $S$ (resp., $C$ and $TP$)

The protocols in [HILM02, YZW06], for example, follow the structure of DR protocols, satisfy the unique-query and query-correctness properties, and either satisfy or can be modified so as to satisfy Property 1 or Property 2.

### 4.2.3 Properties of DPC protocols

We now turn our attention to DPC protocols that can be composed with DR protocols in the manner depicted in Figure 4.2. After the DR setup subprotocol, instead of a single query message Q sent from $C$ to $TP$, we now have three subprotocols (a query subprotocol, a compliance-verification subprotocol, and a query rewriting subprotocol) after which a query message is sent to $TP$, and then the answer step of the DR protocol can be executed. Here, we only formalize the DPC protocol and its requirements (informally described and motivated in Section 4.1), with the understanding that it is intended to be composed with a DR protocol.

The inputs to a DPC protocol are a security parameter $1^\sigma$ (known to all parties), an attribute name and query value $v$ (private inputs to $C$), and a database DB (schema known to all parties, but contents private to $S$). The outputs of a DPC protocol are a query message $Q'$ (communicated privately to $TP$) and a bit $b$ (communicated privately to $S$) indicating whether the query complies with the policy ($b = 1$) or not ($b = 0$).

*Preliminary Requirement Notations:* Let $\sigma$ be a security parameter. Two distribution ensembles $\{D^0_\sigma : \sigma \in \mathcal{N}\}$ and $\{D^1_\sigma : \sigma \in \mathcal{N}\}$ are *computationally indistinguishable* if for any efficient algorithm $A$, the quantity $|\text{Prob}[\, x \leftarrow D^0_\sigma : A(x) = 1\,] - \text{Prob}[\, x \leftarrow D^1_\sigma : A(x) = 1\,]|$ is negligible in $\sigma$ (i.e., no efficient algorithm can distinguish if a random sample came from one distribution or the other). A participant's *view* in a protocol (or a set of protocols) is the distribution of the sequence of messages, inputs and internal random coins seen by the participant while running the protocol (or the set of protocols).

The security and efficiency properties added to the DR protocol by the DPC protocol are as follows.

1. *Preservation of Query Correctness:* for any database DB, policy $p$, query value $v$, and policy values $w_1, \ldots, w_c$, if policy $p$ evaluates to 1 on input $v, w_1, \ldots, w_c$, then except with negligible probability, the set of records received by $C$ at the end of an execution of the DR protocol on input DB and $v_1, \ldots, v_d$, is equal to the set of records received by $C$ at the end of the composition of the DPC and DR protocols on input DB, $v_1, \ldots, v_d$, $p$ and $w_1, \ldots, w_c$.

2. *Compliance Completeness:* for any database DB, policy $p$, query value $v$, and policy values $w_1, \ldots, w_c$, if policy $p$ evaluates to 1 (resp., 0) on input such query value and policy values, then the probability that at the end of the DPC protocol, $S$ outputs $b = 1$, is 1 (resp., negligible in the security parameter $\sigma$).

3. *Compliance Soundness:* for any database DB, policy $p$, and policy values $w_1, \ldots, w_c$, for any efficient client $C'$ participating in the DPC protocol, the conjunction of the following three events only happens with negligible (in $\sigma$) probability at the end of the DPC protocol execution: $S$ outputs $b = 1$, $TP$ outputs a query message $Q'$ with associated query value $v$ and policy $p$ evaluates to 0 on input $v, w_1, \ldots, w_c$.

4. *Privacy:* To model a realistic scenario of multiple database policy compliance protocol executions composed with multiple database retrieval protocol executions, we consider multiple runs of the DPC protocols and augment the adversary and simulator with auxiliary inputs. We define privacy against a malicious adversary by a natural adaptation of the real/ideal security framework commonly used in the cryptography literature, namely: For any efficient (i.e., probabilistic polynomial time) auxiliary-input adversary $Adv$ corrupting one of the three parties (i.e., client $C$, server $S$ or third party $TP$), there exists an efficient auxiliary-input algorithm $Sim$ (called the *simulator*), such that $Adv$'s view and the output of the remaining two parties in the "real world" are computationally indistinguishable from $Sim$'s output and the output of the remaining two parties in the "ideal world", where these two worlds are defined as follows. In the *real world*, multiple runs of the DPC protocol are executed, while $Adv$ acts as the corrupted party. In the *ideal world*, each run of the DPC protocol is replaced with an 'ideal execution' that is designed so to only reveal some 'minimal information', such as system parameters and inputs/outputs based on the policy compliance functionality. On the input of a query value $v$ given by $C$, a database $DB$, policy values $w_1, \ldots, w_c$, and policy $p$ input by $S$, an ideal execution of a single DPC protocol returns:

   (a) the output $b$ of policy $p$ on input query value $v$ and policy values $w_1, \ldots, w_c$ to $S$

   (b) a random query message $Q'$ to $TP$, where $Q'$ has no matching records if

$b = 0$ or has associated query value $v$ if $b = 1$.

In our first two protocols, we also admit some additional leakage to $S$, and thus consider the appropriate variant of the above definition, where such leakage is also admitted in the output of the ideal functionality.

5. *Efficiency:* A DPC protocol's *latency* is measured as the time taken from the time the parties enter their input to the time all parties receive their outputs (as a function of $\sigma$ and other system parameters). The DPC protocol's *communication complexity* (resp., *round complexity*) is defined as the length (resp., number) of the messages, as a function of $\sigma$ and other system parameters, exchanged by $C$, $S$ and $TP$. Even if our analysis mainly focuses on latency, our design targets minimizing all these efficiency metrics.

Our design also targets a number of additional security properties, which can be obtained using known techniques: *confidentiality* of the communication between all participants, message *sender authentication*, message *receiver authentication*, and *communication integrity* protection. These can be realized by using network security protocols such as TLS.

## 4.3 Protocols for Black/Whitelist Policies for Keyword Queries

In this section we present our three DPC protocols (whose properties are detailed in Figure 4.3). Our first protocol $\pi_1$ falls short of satisfying all requirements formulated in Section 4.2 in two ways: (a) it does not satisfy compliance soundness (i.e., a malicious client $C$ could send inconsistent encryptions for compliance verification and query rewriting; thus, the compliance verification test would pass on a query value different than the one used for query rewriting); (b) the privacy against $S$ is only satisfied if the protocol is allowed to leak any repeated occurrences of the same query value. Our second protocol $\pi_2$ extends $\pi_1$ so to eliminate (a), and protocol $\pi_3$ addresses both (a) and (b).

In the following protocol descriptions, keys are named with two subscripts indicating by which parties they are shared. For example, a private key shared by $C$ and $TP$

would be named $k_{\mathrm{c,tp}}$. There may optionally be a third subscript *com* or *que*, to indicate whether the key is used for policy compliance checking or query fulfillment. Thus, $k_{\mathrm{c,s,com}}$ means a key shared by $C$ and $S$ and used for compliance checking. We assume a standard 2-party key agreement protocol, secure against a malicious adversary, is executed in an initialization phase to produce these keys.

### 4.3.1 Basic Keyword Protocol $\pi_1$

Our most basic protocol $\pi_1$ allows efficient enforcement of policy compliance for keyword search queries with whitelist or blacklist policies. (We continue our description for whitelist policies only; blacklist policies can be supported with minor modifications.)

A pictorial description of the protocol can be found in Figure 4.4. In the first step, $C$ sends two double encryptions of its query value $v$, once using key $k_{\mathrm{c,s,que}}$ as the inner layer, and then using key $k_{\mathrm{c,s,com}}$. Both resulting encryptions are sent to $S$.

Then, $S$ and $TP$ interact to analogously compute ciphertexts for the policy values, as follows: first, $S$ encrypts each of the policy values $w_1, \ldots, w_c$ using key $k_{\mathrm{c,s,com}}$ and sends the resulting ciphertexts to $TP$; then, $TP$ further encrypts each of these ciphertexts using key $k_{\mathrm{c,tp}}$, and returns the resulting ciphertexts, *reordered using a random permutation* $\pi$, to $S$. At this point, $S$ computes the whitelist policy output by simply checking whether one or zero of the policy value ciphertexts is equal to the ciphertext received by $C$.

After the policy compliance calculation, if the query is compliant, $S$ simply forwards the received encryption $k_{\mathrm{c,s,que}}$ to $TP$, who can remove the outer layer of encryption and fulfill the query. Otherwise, $S$ performs *query rewriting*, sending $TP$ a random value indistinguishable from a double-encrypted query.

As described in the introduction, the two main technical ideas in this protocol are:

1. using "equality-preserving encryption" to allow $S$ to calculate the policy output without revealing the policy values to $TP$ or $C$ and without learning why the policy was or was not satisfied (i.e., which policy value(s) $w_{j,i}$ may have textually matched value(s) in the query);

2. using "query rewriting" to allow $S$ to rewrite the query $q$ obtained by $C$ into a

query $q'$ which guarantees that the same database records match $q$ and $q'$ if $q$ is compliant, or no records match $q'$ otherwise, without $TP$ or $C$ obtaining any additional information on which is the case.

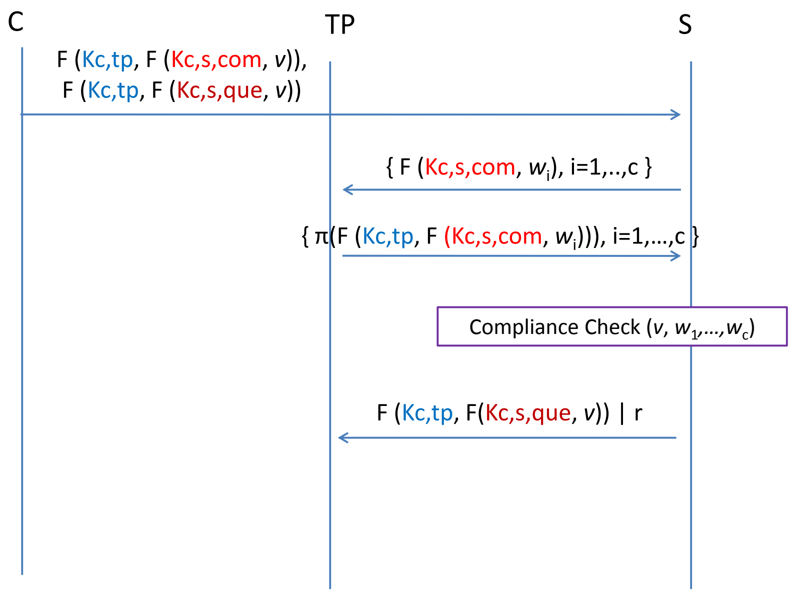

C
TP
S
F (Kc,tp, F (Kc,s,com, v)),
F (Kc,tp, F (Kc,s,que, v))
{ F (Kc,s,com, w_i), i=1,..,c }
{ π(F (Kc,tp, F (Kc,s,com, w_i))), i=1,…,c }
Compliance Check (v, w_1,…,w_c)
F (Kc,tp, F(Kc,s,que, v)) | r

Figure 4.4: The basic keyword search policy compliance protocol $\pi_1$

### 4.3.2 Proof Sketch for Protocol $\pi_1$

*Preservation of query correctness.* We consider a DR protocol satisfying Property 1, and assume that policy $p$ returns 1 on input a query value $v$ from $C$ and policy values $w_1, \ldots, w_c$ from $S$. The set $r_{\mathrm{dr}}$ of records received by $C$ at the end of an execution of the DR protocol on input DB and $v$ is the set of records from DB that contain $v$ at the queried attribute. Now, consider the set $r_{\mathrm{dpc,dr}}$ of records received by $C$ at the end of a composition of the DPC and DR protocols on input DB, $v_1, \ldots, v_d$, $p$ and $w_1, \ldots, w_c$. We observe that $r_{\mathrm{dpc,dr}} = r_{\mathrm{dr}}$ holds when the policy $p$ is satisfied, since the query message $q'$ sent from $S$ to $TP$ during the DPC protocol allows $TP$ to compute the query message $q$ sent from $C$ to $S$ during the DR protocol, by decrypting $q'$ via key $k_{\mathrm{c,tp}}$.

*Compliance completeness.* Assume policy $p$ evaluates to 1 on input a query value $v$

from $C$ and policy values $w_1, \ldots, w_c$ from $S$, and consider first the case where $p$ is the whitelist policy. In this case, we have that $v = w_j$ for some $j \in \{1, \ldots, c\}$. By inspection of the DPC protocol, we see that the ciphertext $F(k_{\text{c,tp}}, F(k_{\text{c,s,com}}, v))$ is equal to the ciphertext $F(k_{\text{c,tp}}, F(k_{\text{c,s,com}}, w_j))$, by properties of $F$, and therefore $S$ outputs $b = 1$. The case where $p$ is the blacklist policy is similarly derived by using the fact that in this case, we have that $v \neq w_j$ for all $j \in \{1, \ldots, c\}$, and the ciphertext $F(k_{\text{c,tp}}, F(k_{\text{c,s,com}}, v))$ will also be different from all ciphertexts $F(k_{\text{c,tp}}, F(k_{\text{c,s,com}}, w_j))$, by properties of $F$, for $j = 1, \ldots, c$, and therefore $S$ outputs $b = 1$. The case where policy $p$ evaluates to 0 is similarly derived, again by using the fact that $F$ is a pseudo-random permutation.

The proof of the privacy property is split into 3 parts, depending on who among $C$, $S$ or $TP$ the adversary corrupts.

*Privacy against $C$.* We assume $Adv$ corrupts $C$ and we exhibit an efficient simulator $Sim$ in the ideal world. Note that $C$ only sends one message to $S$, and this message is based on $C$'s private input. Although $C$ might change its input, $Sim$ can extract it by decrypting $C$'s query message using the shared keys obtained at the end of the key agreement protocol. The rest of the protocol is simulated by running the honest parties' programs.

*Privacy against $S$.* We provide a proof for a single execution of a query; the extension to the case of many queries is obtained by essentially repeating the simulator's step for each single query execution, with only the following difference: repetitions of the same query values are revealed to $S$ and can therefore be used by the simulator to accordingly simulate the message from $C$ to $S$. We assume $Adv$ corrupts $S$ and we exhibit an efficient simulator $Sim$ in the ideal world, so that $Sim$'s output and $Adv$'s view while posing as $S$ in the real world (where a real execution of $\pi_1$ happens) are computationally indistinguishable. In this case, the simulator $Sim$ sends inputs DB and $v$ to the (ideal world) trusted party, and receive as output for $TP$ a query message $q'$, and as output for $S$ a compliance result bit $b$. Then $Sim$ simulates the 4 messages received or sent by $S$ as follows:

1. the message from $C$ to $S$ as $(qmes'_1, qmes'_2)$, where $qmes'_1, qmes'_2$ are uniformly chosen from the pseudo-random permutation domain $\{0,1\}^\ell$
2. the 1st message from $S$ to $TP$ by simply running $S$ as a black box;

3. the message from *TP* to *S* as a sequence of $c$ strings uniformly and independently distributed in $\{0,1\}^\ell$, with the further step in the case $b = 1$ that one of these strings, chosen at random, is set $= qmes'_1$

4. the 2nd message from *S* to *TP* by simply running *S* as a black box.

The proof that *Adv*'s view in a real execution of $\pi_1$ is computationally indistinguishable from *Sim*'s output follows by a standard hybrid argument [GM84], using the following observations. First, we note that the two strings in the message from $C$ to $S$, are computed as outputs of the pseudo-random function $F(k_{\text{c,tp}}, \cdot)$ on distinct inputs, and are therefore indistinguishable to *Adv* from random strings of the same length, as in *Sim*'s output, by the assumption on $F$. Then, we note that $c$ strings in the message from $S$ to *TP*, are simulated by *Sim* exactly as they are computed by $S$ in the real world. Now, we consider the $c$ strings in the message from *TP* to $S$. These are computed as outputs of $F(k_{\text{c,tp}}, \cdot)$ on input the strings in the message from $S$ to *TP*, which are all distinct among them and from previous inputs, with only one possible exception: when $b = 1$, one of these inputs may be a value $F(k_{\text{c,tp}}, w_j)$ equal to the value $F(k_{\text{c,tp}}, v)$ used as input in the first string in the message from $C$ to $S$. In *Sim*'s output, they are simulated as random $\ell$-bit strings with the only exception of setting one of them, chosen at a random position, as equal to the first string in the message from $C$ to $S$. We note that the position of this value is random also in the real world, as *TP* randomly permutes its $c$ strings before sending them. Finally, the 2nd message from $S$ to *TP* is simulated by *Sim* exactly as it is computed by $S$ in the real world.

*Privacy against TP.* We assume *Adv* corrupts *TP* and we exhibit an efficient simulator *Sim* in the ideal world, so that *Sim*'s output and *Adv*'s view while posing as $S$ in the real world (where a real execution of $\pi_1$ happens) are computationally indistinguishable. In this case, the simulator *Sim* sends inputs DB and $v$ to the (ideal world) trusted party, and receives as output for *TP* a query message $q'$, and as output for $S$ a compliance result bit $b$. Then *Sim* simulates the 3 messages received or sent by *TP* as follows:

1. the 1st message from $S$ to *TP* as a sequence of $c$ strings uniformly and independently distributed in $\{0,1\}^\ell$

2. the message from *TP* to *S* by simply running *TP* as a black box;

3. the 2nd message from *S* to *TP* as $q'$.

The proof that *Adv*'s view in a real execution of $\pi_1$ is computationally indistinguishable from *Sim*'s output follows by a standard hybrid argument [GM84], using the following observations. First, we note that the $c$ strings in the message from *S* to *TP*, are computed as outputs of the pseudo-random function $F(k_{\text{c,s}}, \cdot)$ on distinct inputs, and are therefore indistinguishable from random strings of the same length, as in *Sim*'s output, by the assumption on $F$. Then, we note that $c$ strings in the message from *TP* to *S*, are simulated by *Sim* exactly as they are computed by *TP* in the real world. Finally, the simulation of the 2nd message from *S* to *TP* is directly derived from the trusted party's output and is therefore distributed exactly as in the real world.

### 4.3.3 Protocol $\pi_2$: soundness against malicious clients

One problem with the protocol $\pi_1$ described in Section 4.3.1 is that a malicious $C$ can simply send two different queries for compliance verification and query rewriting, thus violating the soundness property. We now describe a protocol $\pi_2$ that prevents this attack with minimal modifications from $\pi_1$.

As a preliminary observation, we see that after the first query message has been sent by $C$, there is no further outgoing communication from $C$. Thus, the only opportunity for $C$ to provide malicious input is before the compliance verification subprotocol. This naturally leads us to examine ways in which we could modify protocol $\pi_1$ to require only one input from $C$. Note that we cannot use the same encryption for both compliance checking and query fulfillment, since that would allow *TP* to identify encrypted query values that match policy items it has seen during the setup phase.

We can resolve this by storing a triple encryption of each database value, per Added DR Property 2. Recall that in protocol $\pi_1$, *TP* holds a singly-encrypted database; after the compliance check, *TP* receives a double-encrypted value $F(k_{\text{c,tp}}, F(k_{\text{c,s,que}}, v))$, and removes the outer layer of encryption to get $F(k_{\text{c,s,que}}, v)$ with which to answer the query. The second layer was needed to prevent $S$ from decrypting the query value. In protocol $\pi_2$, we assume that triply-encrypted values $F(k_{\text{c,s}}, F(k_{\text{c,tp}}, F(k_{\text{c,s}}, v)))$ are used within the DR protocol. We discard the subscripts *que* and *com* as they are no longer relevant.

The structure of the protocol is similar to that of Section 4.3.1. At query time, $C$ encrypts the query value with *both* of its keys and sends the resulting doubly-encrypted value $F(k_{\mathrm{c,tp}}, F(k_{\mathrm{c,s}}, v))$ to $S$. Then $S$ encrypts each of the policy values $w_i$ using key $k_{\mathrm{c,s}}$ and sends them to $TP$, which then re-encrypts each of these using key $k_{\mathrm{c,tp}}$, randomly permutes the order of keywords, and returns the re-encrypted values to $S$.

As before, $S$ checks the encrypted query for equality with the double-encrypted policy values. If the query is non-compliant, $S$ sends to $TP$ a random query indistinguishable from a triple-encrypted real query; otherwise, $S$ re-encrypts $F(k_{\mathrm{c,tp}}, F(k_{\mathrm{c,s}}, v))$ using $k_{\mathrm{c,s}}$, and sends the triple-encrypted value $F(k_{\mathrm{c,s}}, F(k_{\mathrm{c,tp}}, F(k_{\mathrm{c,s}}, v)))$ to $TP$ for its answer generation in the DR protocol. Note that the outermost layer of encryption prevents $TP$ from identifying whether the query matches policy items it had previously encrypted from $S$—thus eliminating the need for separate *com* and *que* encryptions.

The resulting DPC protocol $\pi_2$ inherits the same properties as $\pi_1$, plus compliance soundness under a different assumption on the method used to encrypt the database values in the DR protocol (namely, Added Property 2). (See Figure 4.3 for a detailed property description.) A proof of the properties of $\pi_2$ can be obtained by combining the above observations with the proof of $\pi_1$'s properties from Section 4.3.1.

The triply-encrypted database of Added DR Property 2 can be generated during the setup phase as follows. First, $S$ encrypts all items in the database with $k_{\mathrm{c,s}}$ and sends them to $TP$, which re-encrypts them using $k_{\mathrm{c,tp}}$ and returns them to $S$. Then $S$ adds a third layer of encryption, again using $k_{\mathrm{c,s}}$, and sends the entire triply-encrypted database to $TP$. This interaction between $S$ and $TP$ may be expensive, as it involves every item in the database being both encrypted and sent over the network three times; this may render this method somewhat undesirable to practitioners, especially when dealing with large databases. We address this issue as well in $\pi_3$.

### 4.3.4 Protocol $\pi_3$: privacy across multiple queries

One remaining issue with both protocols $\pi_1, \pi_2$ is the presence of some leakage to $S$ across multiple query executions: $S$ learns, by checking for repetitions in the first message sent by $C$ to $S$, whether the query value in the current execution is equal to a previously executed query. Although not a major form of leakage, it remains of interest to see if we can prevent it at some affordable efficiency cost.

We now describe a protocol $\pi_3$ that keeps all properties in $\pi_1$, the compliance soundness property achieved in $\pi_2$ and satisfies privacy against $S$ without the mentioned leakage. (It also avoids the database setup inefficiency mentioned at the end of the description of $\pi_2$.)

Protocol $\pi_3$ uses an additional cryptographic tool: two-party Secure Function Evaluation (SFE) protocols [Yao86]. Recall that in such protocols, two parties $P_1$ and $P_2$, with private inputs $x$ and $y$, respectively, can jointly compute a functionality $f(x, y) = (f_1, f_2)$, such that $P_1$ receives $f_1(x, y)$, and $P_2$ receives $f_2(x, y)$, and it is required that nothing is learned by either party other than the output.

Instead of using a triple encryption as in $\pi_2$, protocol $\pi_3$ uses a different shared key $k'_{\mathrm{c,s}}$ for query rewriting. After the policy check, which remains unchanged, $S$ and $TP$ perform a two-party SFE protocol, the result of which will be a newly-encrypted form of the query which will be released to $TP$.

First, $C$ sends $F(k_{\mathrm{c,tp}}, F(k_{\mathrm{c,s}}, v))$ to $S$, at which point the policy compliance check, which remains unchanged from Section 4.3.1, takes place. After the compliance check, $S$ and $TP$ engage in a two-party SFE protocol, where $S$ inputs $F(k_{\mathrm{c,tp}}, F(k_{\mathrm{c,s}}, v)), k_{\mathrm{c,s}}, k'_{\mathrm{c,s}}$ a random query $r$, and the (one-bit) result of the compliance check. $TP$ inputs $k_{\mathrm{c,tp}}$. Together, the two parties securely compute the following output, which is received only by $TP$: if the query was non-compliant, random value $r$ is output; if the query was compliant, the doubly-encrypted value $F(k_{\mathrm{c,tp}}, F(k_{\mathrm{c,s}}, v))$ is decrypted twice to produce $v$, which is then encrypted using key $k'_{\mathrm{c,s}}$, and the result, $F(k'_{\mathrm{c,s}}, v)$ is released to $TP$. $TP$ then proceeds to compute the answer based on the DR protocol. The resulting DPC protocol $\pi_3$ is illustrated in Figure 4.5.

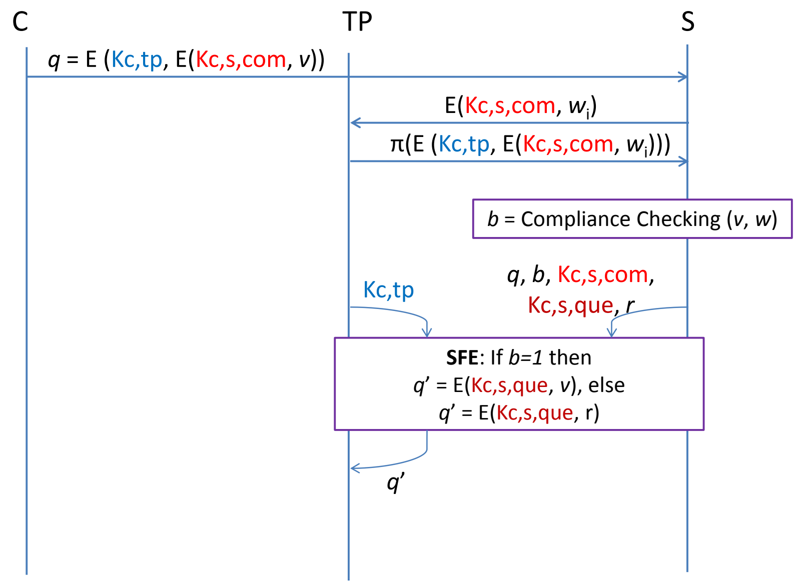

C
TP
S
q = E (Kc,tp, E(Kc,s,com, v))
E(Kc,s,com, w_i)
π(E (Kc,tp, E(Kc,s,com, w_i)))
b = Compliance Checking (v, w)
Kc,tp
q, b, Kc,s,com, Kc,s,que, r
SFE: If b=1 then q' = E(Kc,s,que, v), else q' = E(Kc,s,que, r)
q'

Figure 4.5: Protocol $\pi_3$: Keyword search policy compliance with (multi-query) security against $S$

## 4.3.5 Remarks on Query Rewriting

Each of the protocols above describes, in addition to the compliance checking step, how the server $S$ selects the query to forward to $TP$ for execution. In all three protocols, the Client's query message consists of two parts. The first contains the query to be used for compliance checking, while the second contains the query to be executed by $TP$. For compliant queries, $S$ simply extracts the second part of the original query message from $C$ and forwards it to the Third Party. For non-compliant queries, it extracts the second part of the original Client query message and replaces ciphertext attribute values with random values before forwarding it to the Third Party.

The encrypted database stored at $TP$ is constructed so that every query, including random queries that match zero records, returns a data structure equal to the size of at least one record. This further obscures the distinction between compliant and non-compliant queries in the view of $TP$.

## 4.4 Extensions to Range Query Types and Boolean and Range Policies

### 4.4.1 Boolean Policies

The whitelist and blacklist policies of protocols $\pi_1$–$\pi_3$ can be viewed as Boolean formulas over equality statements. Allowed and disallowed keyword policies can be written as, respectively:

$$\bigvee_{i=1}^{t}(v = w_i) \tag{4.1}$$

$$\bigwedge_{i=1}^{t}\neg(v = w_i) \tag{4.2}$$

That is, a whitelist policy is a disjunction of equality statements: *at least one* of the whitelisted terms must be *equal* to the query. A disallowed keyword policy is a conjunction of inequalities: it must hold that *all* the blacklisted terms are *not equal* to the query term. Each ciphertext encrypting a required (disallowed) keyword is interpreted as an equality (inequality) statement leaf.

This leads us to examine whether more sophisticated Boolean policy structures, coupled with compound queries consisting of more than one keyword, can also be handled in this equality-checking framework.

It turns out that a direct use of protocol $\pi_1$ to check policies with more structure than a flat list will reveal unacceptable additional information about the query to S. Consider the following example CNF policy:

```
(NOT fName = 'John' OR NOT lName='Smith')
  AND (NOT fName = 'Casey' OR NOT lName='Jones')
```

Note that this policy applies only to compound queries in which at least two keyword searches are specified. The goal of this policy type is to forbid two specific combinations of first and last names. Keeping the same encoding as above, where each equality statement in the policy is a single ciphertext, we see that the third party can no longer randomly permute all ciphertexts, as equality statements must remain associated with the clause in which they appear to determine the formula's truth value.

We can resolve this by sending the ciphertexts to $TP$ for re-encryption in separate batches for each clause, and the third party will permute *both* the ciphertexts within a clause and the order of clauses. In this case, the third party learns slightly more about the structure of the policy; namely, how many clauses it has and the arity of each clause. Depending on the scenario, this may be an acceptable leakage.

More serious, however, is that in the case of compound queries, $S$ may learn additional information about the query by a partial match. For example, if C's query is `SELECT * FROM T WHERE fName='John' and lname='Jones'`, the server will learn that both of the query items match policy items, even though the query remains compliant. In general, if each keyword search in a query is encoded as a separate ciphertext, a policy may match more than one query ciphertext, revealing additional information about the query beyond what is implied by the policy and the 0/1 compliance result. We see that the *unique match property* is crucial to the security of techniques such as the ones used in this chapter, based on equality-preserving encryption.

We can solve this problem for conjunctive queries and some policy types by means of *tupling*. Consider prohibitive CNF policies such as the example above, where each clause has a fixed arity $a$. By "prohibitive" we mean CNF's over negated equalities only. Clauses such as `(NOT fName = 'John' OR NOT lName='Smith')` can be rewritten as `NOT (fName = 'John' AND lName = 'Smith')`. Now the $a$ conjoined keywords/attribute names within a single policy clause can be encrypted together, so that there is a single ciphertext block for each clause.

The client sends ciphertexts for *every tuple* of keyword searches in the query, for every arity from 1 to a predetermined maximum arity, with each tuple encrypted as a single block, with the keywords in a tuple specified in a canonical (sorted) order. Since the client always sends all possible tuples of each arity up to some fixed constant, she learns nothing about the form of the policy. We assume it is permissible for the server to learn only the *number* of keyword searches contained in the query.

Now the equality checking required to determine policy compliance can be done on entire encrypted clauses. We are free from leakage to $S$ due to multiple matches, as long as each clause in the policy applies to the same set of attributes, and each query searches on no more than one keyword per attribute. By tupling, we have effectively reduced the problem to the single-keyword disallowed keyword case, restoring the 0/1 match property essential for query privacy. Dually, the tupling technique also allows secure checking of permissive DNF policies, that is, ORs of ANDs of equality statements.

### 4.4.2 Range Queries and Policies

Being restricted to policy checking by equality testing of ciphertexts does not, in fact, limit us to keyword policies over keyword queries. In this section we give brief remarks illustrating how the method may be generalized to range queries and policies.

A *range query* is a query of the form `SELECT * FROM T WHERE <attribute> <op> <value>`, where `<op>` is one of $>$, $<$, $<=$, or $>=$, or the two-sided version `SELECT * FROM T WHERE <attribute> BETWEEN <value1> AND <value2>`.

A *range policy* may take the form of a query range *size restriction* or query range *limits*. A range size restriction policy, for example `RANGESIZE age < 35`, applies only to range queries, while a range limit policy, such as `age IN [18,35]`, applies to both keyword and range queries.

A basic technique for approximate enforcement of range-related policies is *binning*. In this approach, the domain of each attribute in the database is divided into a constant number $K$ of equal-size bins, where the domain size and $K$ are both known to $C$ and $S$. To encode a range limit policy, which prohibits queries in a subset of bins, $S$ takes all subsets of the $K$ bins that include one or more of the prohibited bins, and encodes these subsets as integers in $[0, 2^K - 1]$. For a range size restriction policy, all subsets of bins spanning a larger-than-legal range are encoded. These are the policy items ($w_i$) to be doubly-encrypted and permuted for the compliance check, as in the paper's main protocol.

At query time, $C$ sends $S$ the double encryption (with $k_{\mathrm{c,tp}}$ and $k_{\mathrm{c,s}}$) of the integer representation of the subset of bins covered by the query (if the query is a single keyword, it will be in a single bin; a range query may involve multiple bins.) If the query overlaps any disallowed bin or bins, or spans a larger range of bins than is allowed, this encoded subset will match exactly one of the policy ciphertexts; otherwise it will match none.

The limitation of this method is that the expressible policies are restricted by the small number of fixed-size bins, since this encoding requires us to represent and transmit all subsets of the bins. If the desired range policy limits are not on bin boundaries, false positives can result: while no query forbidden by the policy will be allowed, queries that fall in an allowed range but within the bounds of a disallowed bin will be rejected.

A more advanced variant, which overcomes the false-positive problem of binning, is to use a *compact gap* representation of ranges. This is to be treated more fully in an upcoming work.

## 4.5 Performance Measurements

In this section we report various performance results related to implementations of our basic DPC protocols. We focus on results for $\pi_1$ as the performance of $\pi_2, \pi_3$ is similar. (Specifically, $\pi_2$ is only slower than $\pi_1$ by a small constant multiplicative factor and $\pi_3$ is only slower than $\pi_2$ by a small constant additive factor.)

**Setup.** The Server and Third Party processes were running on a Dell PowerEdge R710 server with two Intel Xeon X5650 2.66Ghz processors and 48GB of memory. The R710 server was running 64-bit Ubuntu 12.04.1 and was connected to a Dell PowerVault MD1200 disk array containing 12 2TB 7.2K RPM SAS drives arranged in a RAID6 configuration. The Client was running on a Dell PowerEdge R810 server with two Intel Xeon E7-4870 2.40GHz processors and 64 GB of memory. The R810 server was running 64-bit Red Hat Enterprise Linux Server release 6.3 and was connected to the the Dell PowerEdge R710 server via switched Gigabit Ethernet. The database schema used for the experiments here reported is shown below in the form of SQL DDL.

```
CREATE TABLE main (id BIGINT NOT NULL, fname VARCHAR(11), lname VARCHAR(15),
          ssn CHAR(9), dob DATE, address VARCHAR(100), city VARCHAR(35),
          state VARCHAR(52), zip CHAR(5), sex VARCHAR(6), race VARCHAR(18),
          marital_status VARCHAR(14), school_enrolled VARCHAR(32),
          citizenship VARCHAR(28), income INTEGER, military_service VARCHAR(20),
          language VARCHAR(32), hours_worked_per_week INTEGER,
          weeks_worked_last_year INTEGER, notes1 TEXT, notes2 TEXT,
          notes3 VARCHAR(250), notes4 VARCHAR(50), fingerprint BLOB);
```

The database was populated by generating random values about fictitious people using demographic information from the US Census Bureau. We used the following query templates and compliance policies.

| **Query** | **Template** |
|---|---|
| *Q1* | SELECT * FROM main WHERE lname=value |
| *Q2* | SELECT * FROM main WHERE state=value AND lname=value AND zip=value |

| **Policy** | **Compliant queries must include:** |
|---|---|
| *F* | All queries are rejected as non compliant |
| *T* | All queries are accepted as compliant |
| *B1* | A conjunction of at least 3 keyword queries on *state*, *lname*, and *zip* |
| *B2* | A conjunction of at least 3 keyword queries on *state*, *lname*, and any one of the remaining columns, excluding $fingerprint$ |
| *W1* | A keyword query on *lname* with query value in a 1-entry whitelist |
| *W2* | A keyword query on *lname* with query value in a 100-entry whitelist |
| *W3* | A keyword query on *lname* with query value in a 1000-entry whitelist |
| *W4* | A keyword query on *lname* with query value in a 10000-entry whitelist |
| *W5* | A keyword query on *lname* with query value in a 20000-entry whitelist |

Compliance policy $B2$ was expressed as a disjunction of 23 sub-policies of $B1$ type, each of them requiring keyword query conjunctions on *state*, *lname*, and an additional (and different) database column.

**Results.** Each query template was executed several times using different values. We note that policies $F$, $T$, $B1$ and $B2$ only refer to the query structure and do not depend on query values, contrarily to queries $W1, \ldots, W5$, which depend on values in the query and in (variable-length) whitelist.

Figure 4.6 shows computation results when checking compliance for policies $F$, $T$, $B1$, and $B2$ against instances of the two query classes $Q1, Q2$. Note that checking compliance for $Q1, Q2$ only required checking the query structure and did not require running $\pi_1$, as the policies do not depend on actual query values. Thus, there is no impact from cryptographic operations on the measured running time. Two main observations can be derived from this figure: (1) running time is essentially linear with policy size (e.g., the ratio of the time taken for policy $B2$ to the time taken for policy $B1$ is about the same ratio of the size of $B2$ to the size of $B1$); (2) running time is almost the same for the two policy types $Q1$ and $Q2$, with variance of less than 3%.

Figure 4.7 shows computation results when running protocol $\pi_1$ on policies $W1, \ldots, W5$, against instances of the two query classes $Q1, Q2$. These policies do depend on query values and thus trigger an execution of protocol $\pi_1$, with its cryptographic operations. The main observation derived from this figure is that the time required by $\pi_1$ grows linearly with the size of the whitelist, which can be explained as follows. First, note that policies $W1, \ldots, W5$ only differ in the size of the whitelist. Then, note that as the size of the whitelist grows, so does the time it takes to doubly encrypt its values, send/receive them between $S$ and $TP$, and checking by using sequential scan whether an attribute value referenced in $C$'s query belongs to the doubly encrypted and permuted whitelist values. (Here, a speedup due to the use of binary search does not seem to impact the running time substantially, due to the double encryption and network communication required.)

When comparing the two figures, we observe that the impact of running vs not running $\pi_1$ when checking compliance is essentially minimal for policies with short-size whitelists (i.e., a factor of about 10, calculated by comparing the running time for $F, T, B1$ with the running time of policies $W1, W2, W3$).

the results shown in Figure 4.6, whitelist policies are more expensive. In addition, the time required by $\pi_1$ grows linearly with the size of the whitelist.

For all policies, the time (reported in microseconds) required for checking whether the queries were compliant or not was almost the same for the two policies and query classes (maximum variance of less than 3%). As expected, compliance policy checking takes longer when the policy size increases. For example, compliance policy checking for $Q2$ takes jumps from 410.3 microseconds for $B1$ to 5,958.7 microseconds for $B2$.

However, compliance policy checking does not seem to be very sensitive to query sizes. This is due to the way compliance checking is carried out. In particular, for a Boolean compliance policy consisting of one or more disjunctions (ORs) of conjunctive (ANDs) terms, each conjunctive clause appearing in the disjunctive normal form (DNF) representation of the query is checked against policy disjunctions until either a "match" is found or all disjunctions are checked. A match corresponds to a compliant query conjunctive clause. All query conjunctive clauses must be compliant in order for the query to be compliant.

Since both $Q1$ and $Q2$ contain just one conjunctive term, compliance checking for $B1$ involves checking just one conjunctive policy term, while compliance checking for $B2$ involves checking up to 23 conjunctive policy terms. For $Q1$, all 23 conjunctive policy terms must be checked. For $Q2$, the number of checks depends on the position of the matched conjunctive policy term in the list of all conjunctive policy terms. In the results shown in Figure 4.6, it was the tenth entry. Because of this difference, compliance checking for $Q1$ takes 3% more than compliance checking for $Q2$ (6,125.4 vs 5,958.7 microseconds).

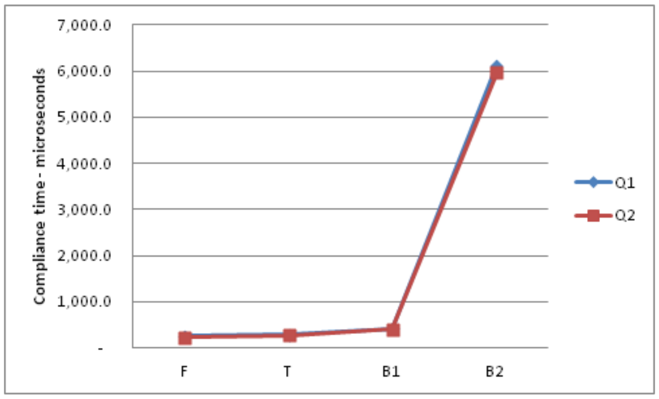


Figure 4.6: Query compliance checking overhead for policies $F$, $T$, $B1$, and $B2$.

For example, running $\pi_1$ for $Q2$ required 646.4, 1,877.5, 10,639,1, 103,190.8, and 200,694.3 $\mu s$ for $W1$, $W2$, $W3$, $W4$, and $W5$, respectively. This is due to two factors. First, whitelist values are first encrypted by $S$ and then sent to $TP$ (over TLS) for an additional encryption and permutation to prevent $S$ to learn which entry matched a

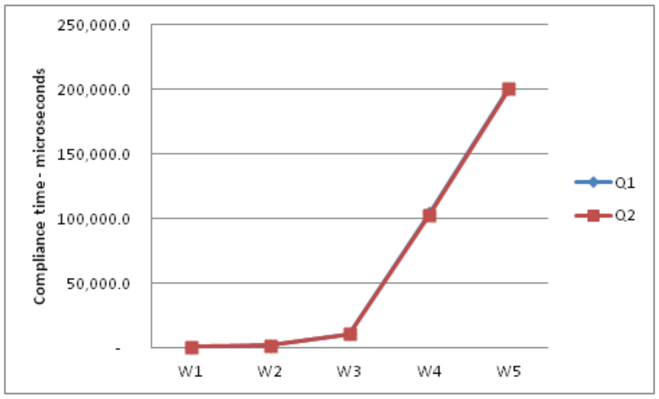


Figure 4.7: Query compliance checking overhead for policies $W1$, $W2$, $W3$, $W4$, and $W5$.

query term. As the size of the whitelist grows, so does the time it takes to doubly encrypt its values and send/receive them between $S$ and $TP$. Second, $S$ checks whether an attribute value referenced in $C$'s query belongs to the doubly encrypted and permuted whitelist values by using sequential scan. As the size of the whitelist grows, so does the number of comparisons required, with non-compliant queries requiring as many comparisons as the size of the list. Obviously, for very large whitelist sizes, binary search would improve substantially the time required for checking whether an attribute value referenced in a query is in the list or not. However, such improvements may not have a large impact on the overall query compliance time due to the double encryption and network communication required. The query rewriting process requires almost the same time for both compliant and non-compliant queries, the latter being slightly most expensive (less than 50 $\mu s$ for $Q2$ queries).

## 4.6 Conclusion

A primary insight from the results in this chapter is how adding non-fully-trusted parties to a distributed computation can enable efficient privacy-preserving protocols—in this case, it enables us to avoid expensive computational assumptions such as the assumptions that enable public-key cryptography. Essentially, we are distributing trust.

Previous work in two-party computations that are assisted by a semi-trusted third party, sometimes referred to as the *commodity-based*, *server-assisted*, or *server-aided* model, seems to have originated in [Bea97], which proposed oblivious transfer protocols in this model. Another three-party protocol for oblivious transfer that achieves unconditional security is also presented in an unpublished manuscript of Rivest [Riv99].

Even with a third party, our protocol has a form of leakage when multiple queries over time are considered. This leakage seems to be intimately related to input/output size issues. General MPC protocols achieve security without leakage by assuming a known, fixed size of inputs and outputs (possibly by padding to a maximum size), and by processing the entire inputs of all parties in every execution of the protocol. For a large database system that answers multiple queries, this is not practical, as 1) the server's input length is the size of the entire database, and 2) the size of a query answer may vary significantly, up to a significant fraction of the database size. The PIR lower bounds of Haitner et al. [HHS08] show that in the two-party setting, some leakage is unavoidable without incurring overhead that is *linear* in the size of the whole database. In a large database, overhead proportional to *any* non-constant function of the database size may be too large to be practical.

Though the protocols presented in this chapter successfully eliminate all dependency on database size by adding a semi-trusted third party, some leakage remains, as described above. The only type of leakage not prevented by any of the protocols $\pi_1$, $\pi_2$, or $\pi_3$ is the third party's observation of output lengths. Having leakage based on variability in output size seems inescapable, and For these specific protocols, we feel that the level of leakage is not serious. Since the third party never sees unencrypted queries or records, it is not clear that anything damaging can be learned by analysis of result lengths. Regarding policy enforcement, since the private fulfillment protocol only hides the distinction between zero-record and one-record results, queries that return an answer of single-record length are more likely to have been non-compliant. This, in turn, allows the third party to gain information on the strictness of the policy relative to the client's query pattern. In the case of multiple clients with a single policy, the third party may gain information about the relative frequency with which clients issue non-compliant queries. In the limit, an observant third party may be able to learn which entries in its encrypted database are in fact bogus records returned as the answer to non-compliant queries. Note that this attack actually becomes more difficult as the size of the database increases relative to the number of queries observed.

This is not to say that all types of leakage that seem innocuous are in fact so. In the next chapter, we examine a case where a leakage profile that was formerly considered to be harmless can be used to mount devastating attacks.

# Chapter 5

# The Practical Security of Searchable Encryption

This chapter deals with protocols for another family of secure computation functionalities for machines in one specific configuration—namely, when there is a single client and a single server to whom the client uploads personal data in the form of documents. The functionality is that the client can search the documents stored at the server, with the server learning as little as possible about the client's data and search terms. This variety of secure computation has similarities to the database protocols of the previous chapter in that it facilitates searching of remotely held private data, but it differs in that it involves only two parties, and only the server is considered a potential adversary, since the data belongs to the client.

In the literature, schemes for accomplishing this efficiently are called Searchable Encryption (SE) schemes. They achieve high efficiency with provable security by means of a quantifiable leakage profile—specifically, leakage of *access patterns*. However, the degree to which SE leakage can be exploited by an adversary has not been well understood. Moreover, in our investigations, we have discovered that deployed security products often have greater leakage than SE. It may be said that searchable encryption technologies have the opposite problem of MPC; while MPC suffers from a low adoption rate in spite of its high provable security level, searchable encryption schemes are being hastily adopted and deployed without a thorough understanding of what security properties they do or do not provide in a given setting.

Our approach to ameliorating this condition has some kinship to the approach for systematizing general MPC presented in Chapter 3. In this chapter we present a characterization of the leakage profiles of in-the-wild searchable encryption products and SE schemes in the literature, and present attack models based on an adversarial server's prior knowledge. Then we empirically investigate the security of searchable encryption by providing query recovery and plaintext recovery attacks that exploit these leakage profiles. We call these attacks *leakage-abuse attacks* and demonstrate their effectiveness for varying leakage profiles and levels of server knowledge, for realistic scenarios. Among our contributions are particularly effective *active attacks*, which have not been previously explored.

## 5.1 Introduction

Encryption protects data stored at an untrusted service provider, but introduces complications. For instance, the service provider is unable to process the encrypted data as freely and efficiently as it can plaintext data, making access cumbersome for the data owner. To address the difficulty of retrieving encrypted text efficiently, increasingly practitioners turn to *searchable encryption (SE)* schemes, first introduced by Song, Wagner, and Perrig [SWP00]. An SE scheme encrypts a set of documents in a way that allows the data owner to delegate the ability to search the documents without decrypting them. For example, using SE, a user may encrypt her email, store it at the remote provider, and later have the provider search the email on a per-keyword basis. If the SE scheme is *dynamic*, then she will also be able to add encrypted documents efficiently.

Many SE constructions can be implemented using only a blockcipher, and some are even legacy-compatible, requiring no modification at the provider. A commonly deployed example of the latter, and one suggested in a number of recent research papers [HAJ$^+$14, LCS$^+$14], is to append to a conventional encryption of the document a sequence of outputs of a secret-keyed pseudorandom function (PRF) on individual keywords. Search is performed by submitting the PRF output of the desired search term, and documents can easily be added or removed. The advantage of such schemes is that they are compatible with existing storage and search indexing systems. Several deployments have been observed in the wild, particularly in settings where encryption

| Objective | Prior Knowledge | Pas/Act | Min Leakage | Known Constructions | Where |
|---|---|---|---|---|---|
| Query Recovery | Fully Known Docs | Passive | L1 | All | ([IKK12], § 5.5.2) |
| Query Recovery | Partially Known Docs | Passive | L1 | All | (§ 5.5.3) |
| Plaintext Recovery | Known Doc Subset | Passive | L3 | – | (§ 5.7.1) |
| Plaintext Recovery | Distributional | Chosen Doc | L3 | – | (§ 5.7.2) |
| Plaintext Recovery | Distributional | Chosen Doc | L2 | Shadowcrypt, Mimesis | (§ 5.7.2) |

Figure 5.1: Summary of successful attacks on searchable encryption schemes. In all cases the adversary sees the transcript of communications to the server, in addition to the indicated knowledge. The leakage levels are defined in Section 5.3.

must be transparent to the user of an application. For example, if data in a cloud service must be encrypted but searchability must also be preserved, an SE scheme is used.

The security provided by SE against an untrusted service provider is, however, inherently weaker than what is achieved by encryption without searching. All efficient SE constructions expose some information, called *leakage*, about the plaintext to the service provider. Typically, SE schemes allow the provider to learn about the underlying data by observing statistics like the number and frequency of encrypted documents accessed during searches. Song et al. recommended periodic re-encryption to address what they call *statistical attacks*, but did not investigate this further. Islam, Kuzu, and Kantarcioglu initiated the study of the empirical security of SE, showing that a user's queries can be guessed with high accuracy when the term distribution of the dataset is known to an honest-but-curious service provider [IKK12]. Their attacks, and the more general statistical attacks alluded to by Song et al, are possible even when a scheme has been proven secure under a standard assumption. In other words, the attacks are permitted by the security definition, and can be seen as the cost payed for efficient retrieval.

The risk posed by leakage has not been closely scrutinized beyond the results of Islam et al. The current literature leaves a practitioner with few concrete recommendations for configuring and deploying SE, and as we show, small variations in details lead to devastating attacks. The risk is not merely abstract; several deployments of SE can be easily broken depending on how they are used.

**Our contributions.** We study the leakage of SE in order to understand its practical security. We expand the work initiated by Islam et al., which considers only a single attack type against standard SE, by considering a range of threats against various approaches to building SE. We enumerate new threat models that describe attacks against suggested SE use cases, like encrypted email, and explore the efficacy of several attacks against different constructions of SE. In each case, we design attacks that exploit the leakage rather than any particular constructions. We term these *leakage abuse attacks* (LAAs), and explore several LAAs against leakage profiles for constructions found in the literature and in use. In some use-cases we show how the details of an implementation may leave SE fatally vulnerable to attacks not previously explored in detail.

**New threats.** We classify multiple threats against SE. Islam et al. studied *query recovery*, where the untrusted server can recover the plaintext terms being searched for. To this we add goals for an adversary which are likely more immediate concerns: An attacker may *recover the plaintext documents themselves* rather than only the queries as they are issued. Perhaps surprisingly, the plaintext recovery has not been treated directly in the SE literature, despite being intuitively more dangerous than simple query recovery.

**New attacks.** We consider attacks against SE schemes admitting different levels of leakage, performing experiments using public email data sets as stand-ins for confidential emails. For a summary of the attack results in this chapter, see the table in Figure 5.1. We first revisit the query recovery attack setting of Islam et al. We show that with common leakage profiles a much simpler and faster attack, that does not require optimization software, is possible.

We also explore, for the first time, *active attacks* where an attacker can induce a user to insert chosen documents, or somehow identify known documents in the corpus. We show that these attacks can easily extract significant information about plaintexts. As one example, using the Enron email dataset for simulations, we show that a server that happens to know a single email sent to 500 Enron employees and that is indexed learns on average 35% of the keywords of all other encrypted emails. These attacks are plausible when SE is used to encrypt email and automatically add email to the encrypted document set – In this case, all the attacker needs to do is send the victim an email, or identify an email that it knows (like an institutional announcement).

In summary, we are the first to investigate the implications in practice of common SE leakage models. Our experimental analyses show that efficient and easy-to-mount attacks can reveal a significant amount of information about email plaintexts, with severe consequences for the security of deployed systems as well as research prototypes [HAJ$^+$14, LCS$^+$14].

We begin in Section 5.2 by discussing the features of multiple currently deployed searchable encryption technologies. In section 5.3 we abstract from these, presenting our classification of searchable encryption schemes. In terms of their operation, the key distinction in SE is between "in-place" schemes, in which encrypted documents are stored on the server directly, and index-based schemes, in which clients upload an encrypted (or unencrypted) index structure to the server. Then we define a set of *leakage profiles* by which the leakage of all schemes we discuss can be classified. Finally, we define *attack models* which describe the types of information that an adversarial server can gain through abusing SE leakage. These characterizations effectively highlight the distinctions and commonalities of SE schemes in a way that provides systematic understanding of their vulnerabilities.

The presentation of statistics from our attack experiments begins in section 5.4. In Section 5.5 we show query-recovery attacks that can be mounted by a server who only passively observes an SE scheme's uploaded data structures. In the case where the document set itself is public (or otherwise largely known to the server), we present a generalization of the attack of Islam et al. [IKK12] for index-based schemes that quickly recover nearly all of the queried terms.

In Section 5.7 we present even more damaging attacks against what we define as in-place schemes, requiring minimal additional knowledge by the server. We cover passive attacks and active *document insertion attacks*, characterizing the tradeoff between length of maliciously planted documents and the adversary's ability to recover plaintext keywords.

We conclude with remarks on the vulnerabilities exposed by this work and the continuing need for empirical investigation of secure protocols that have a leakage component.

## 5.2 Survey of Implemented SE Constructions

In this section we informally describe SE constructions observed "in the wild". We give two categories of constructions. *Legacy-compliant* SE schemes are those that work within an existing application with no modification at the provider end. They work by leveraging the provider's existing information retrieval system without requiring the provider be aware of encryption. The other category is *non-legacy-compliant* schemes, which require special software (though sometimes very simple) at the provider.

**Legacy-compliant solutions.** An example architecture for a legacy-compliant solution is given in Figure 5.2, and reflects the design of industry products that provide searchable encryption to services like Google GMail, Salesforce, and Dropbox. Here, a proxy (which can be a browser plugin or a network proxy) intercepts plaintext documents, and processes and encrypts them before sending to the provider. Later, the proxy intercepts a query and translates it to (possibly several) *encrypted queries* that it forwards to the provider, who processes them normally and produces a response. The proxy uses the responses to provide the plaintext results of the query to the user.

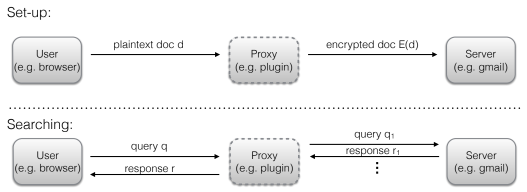


Figure 5.2: Architecture for a legacy-compliant SE scheme.

In this setting, the provider's software is not modified to support encryption. Our architecture above also does not modify the user-facing part of the application, but we will consider modifications at the user side to be legacy-compliant. It is likely cheaper and easier for an individual client to use special encryption software.

Legacy-compliance introduces several challenges. Document storage and searching must be implemented using an existing API that is unaware of encryption. The API provided will typically expose interfaces for document insertion and text search queries, but the index construction and details of the actual computation are hidden from the client software. One might hope to leverage the full power of an IR system, which may perform various types of semantic analysis on the documents. But naive approaches reveal significant plaintext data to the server or incur prohibitive performance penalties.

Consider also that in legacy-compliant schemes, the software issuing the query (usually the proxy) must have the same key used by the client for indexing. This is more of an issue in the browser plugin setting, where all parties loading data into the application must share a common key. A full discussion of key management in these settings is beyond the scope of this dissertation.

**Non-legacy-compliant.** A variety of approaches are available when the provider's software can be modified. These range from highly-secure but less-efficient constructions using FHE to systems that are almost as efficient as their unencrypted counterparts. Our focus will tend towards the latter. Non-legacy compliance opens a wide design space and we will examine various approaches.

In this setting, implementation complexity may be important. An SE system that can utilize parts of existing infrastructure for large-scale document storage, indexing, and searching may be cheaper to deploy and give better performance.

All of the implementations we discuss below are highly efficient, making use of only symmetric-key cryptography (if even that), and the only communication overhead incurred is due to padding each encrypted word to the maximum word length.

### 5.2.1 Shadowcrypt

Shadowcrypt [HAJ$^+$14] is a highly usable, legacy-compliant SE project developed at Berkeley and UMD that aims to protect a user's input into a web form by encrypting the contents of a text field in a way that is transparent to the user. It does this using ShadowDOM, a new standard for building DOM trees (or sub-trees) on top of the existing webpage. To use Shadowcrypt, a user simply clicks a lock next to the field and enters his or her input as usual. Then, the Shadowcrypt plugin encrypts the input before the actual webpage has access to it.

One feature of note in Shadowcrypt is its ability to preserve keyword search on the input to a form. Shadowcrypt accomplishes this by hashing the keywords in the form and appending the hashes to the end of the ciphertext, effectively creating a "bag/set of words" searchable encryption scheme.

We analyze general attacks against Shadowcrypt-like schemes in Section 5.7, but for the sake of concreteness we mention here that a malicious server may obtain a chosen-document oracle by a simple UI redressing attack. First, the server must detect that the user has Shadowcrypt enabled, perhaps by scanning the user's existing documents for random-looking data matching a Shadowcrypt ciphertext. Then, the server sends the user an email or document containing words it would like to see indexed. It also builds the UI in such a way that the "encrypt field" button is hidden behind a field the user must click. In the case of a malicious email server admin, this might be a "send email" button. Then, when the user responds to the email, he or she unwittingly clicks on the hidden button as well, which causes the field's contents to be encrypted. Finally, the server receives the email and adds it to its list of known keyword hashes, which it can use to recover documents or queries.

The authors of Shadowcrypt acknowledge this type of attack is possible, but their concern is more with the prevention of traditional clickjacking attacks. They did not consider the possibility of this kind of attack against their keyword search scheme.

### 5.2.2 Mimesis Aegis

Mimesis Aegis [LCS$^{+}$14] is a project and prototype developed at Georgia Tech which has a similar use case to Shadowcrypt. It aims to preserve user privacy and data confidentiality on mobile platforms by interposing a transparent encryption layer on top of the GUI of mobile applications. The basic idea is similar to Shadowcrypt - encrypt user transparently before it is touched by application code. The same transparent layer then decrypts the data in the application before it is presented to the user. They accomplish this with an adaptation of the operating system's accessibility layer, which allows them to "proxy" incoming user actions. Because they are focused on preserving the user experience as much as possible, they also included a mechanism for enabling search over the encrypted data which they call "easily-deployable efficiently-searchable symmetric encryption", or "EDESE". They discuss several different ways of accomplishing this functionality, but all of them are basically creating a "bag of words" by appending the result of evaluating a PRF on each word in the document to the document's ciphertext. One trick they use reduces the size overhead of the tags by encoding the MACs in a Bloom filter, which trades strict search accuracy for space.

We were not able to obtain a copy of the software for testing, but we postulate that a Shadowcrypt-style UI redressing attack could be used against Mimesis to allow an adversary to index documents of its choosing.

### 5.2.3 Commercial products

Despite the general lack of understanding of the consequences of different types of leakage in searchable encryption, there is a growing market for software designed to preserve search on encrypted data in cloud services. Such software is largely targeted at enterprise customers of SaaS applications who are concerned about security or bound by regulation to encrypt their data at rest. This software usually is deployed as a transparent encryption proxy, fitting the "legacy-compliant" model shown above.

A full comparison of schemes used by these products is beyond the scope of this thesis.

## 5.3 Characterization of SE Schemes and Leakage

In this section we give precise definitions of SE schemes, together with their leakage profiles, abstracted from implementation and deployment details. It will be pointed out which schemes the existing SE products described in the previous section are instances of. Then we describe attack models based on the adversarial server's capability.

First we present some standard notation used in the literature to describe SE schemes, to be used throughout the remainder of the chapter.

**SE basics.** We let $\mathbf{D} = (D_1, ..., D_n)$ denote a collection of $n > 0$ plaintext documents, where each document is a variable-length string from a known character set. We denote the length of a document in characters by $|D_i|$. A keyword extraction procedure KeyExtract takes as input a character string $D_i$ and outputs a vector $W_i$, the keywords, where each component is a character string. A typical keyword extraction will first parse $D_i$ into words, drop common words such as articles and propositions, and stem the remaining individual words. We assume the keyword extraction procedure is deterministic and, looking ahead, known to the adversary. We denote by $c_i$ the count of keywords in, or equivalently the dimension of, $W_i$. We let $\mathbf{W} = (W_1, \ldots, W_n)$ be the set of all the documents' keyword vectors.

An SE scheme consists of encryption, search, and (possibly) update algorithms. The encryption algorithm takes as input a secret key $K$, documents $\mathbf{D}$ and emits a ciphertext. Search takes as input a secret key $K$, a keyword $w$, and outputs a query message. If a scheme includes the update algorithm, it is known as a *dynamic* SE scheme. The update algorithm takes as input $K$ and a document $D$, and outputs an update message. Most published SE schemes are in a non-interactive model in which query and update messages are sent to the server, the latter executing some algorithm and returning a result. The client can then process the result, e.g., decrypting recovered documents due to a search query.

### 5.3.1 A Classification of SE Scheme Types

All of the schemes described here leak information to the server in the form of occurrence patterns of encrypted keywords in documents. This leakage is due either to deterministic encryption of keywords or co-occurrences in an index structure. Security definitions for SE describe the allowed leakage using a *leakage model* that bounds the information an adversary should learn. One proves that an adversary's view during an attack can be simulated given the leakage profile. See the citations given in the next section for representative analyses. These analyses rule out attacks by an adversary that obtain information not captured by the leakage model. However, except for the results of Islam et al., none perform any analysis of how attackers may abuse the leakage.

We developed a scale for characterizing leakage, defining a set of four *leakage profiles.* Below we identify four classes of leakage profiles relevant to our attacks. The leakage profiles often include other pieces of information (see, e.g., Figure 1 of [CJJ$^+$13]) such as, say, the number of documents, or the number of unique keywords, when the same query is repeated, and so on. While possibly relevant in practice, these variations seem inconsequential to our attacks and we ignore them in our treatment. We note, however, that certain statistics like the plaintext lengths, when queries repeat, and even a query's time of day or client IP address may enable simple attacks other than the ones we explore.

The definitions of leakage profiles follow along with the description of schemes that instantiate them. The descriptions progress in order from greatest leakage (L4) to least (L1). We also stress that the same leakage profile may arise for two very different schemes (we note an example below for L2). For the sake of brevity we will typically only informally describe the portions of SE schemes that are not relevant to our results. We do note that all the schemes we will consider are perfectly correct.

#### 5.3.1.1 In-place SE schemes

The first two schemes are called *in-place* because they involve direct uploading of encrypted document data, and the server searches by iterating over keywords on a per-document basis. These are the simplest to implement, but also may have greater leakage than index-based schemes.

**Full-text substitution cipher.** This is the simplest type of searchable encyption. The client parses each input document and performs keyword extraction to produce keywords $W_i$ that are in order with repeats. Then, it applies a deterministic cipher $E$ to each word. The resulting collection of ciphertexts are uploaded. The client searches by sending the enciphered version of a keyword to the server. The server scans the encrypted documents to match the term ciphertext or, alternatively, it could use an inverted index pre-computed over the individual encrypted keywords in the ciphertext.

This scheme enables the server to carry out keyword, boolean, and phrase searches efficiently, and to generate any type of index. However, stemming, wildcard, and approximate-match searching are not possible. This is because the value indexed by the server is a pseudorandom token which has no relation to the plaintext. So, for example, a wildcard search like "s*" would fail because the tokens for keywords starting with "s" do not themselves start with "s".

This simple in-place scheme has the largest leakage of any that we study. We define its leakage profile as:

L4: *Full plaintext under deterministic word-substitution cipher.* The server learns the pattern of locations in the text where each word occurs and its total number of occurrences in the document collection.

We do not present explicit attacks against only this leakage type, though the attacks against all lower leakage profiles also apply.

The solution for searchable encryption described in [SWP00] improves the security of the basic scheme above by additionally using a separate stream cipher for each keyword. The cipher iterates sequentially over each occurrence of the keyword in the document set. Thus, in the uploaded data each occurrence of a keyword has a different ciphertext, and the server cannot observe the pattern of repeated keywords. To search, the client sends a ciphertext corresponding to the first location of the search term, and the server iterates the cipher to find all word matches in the document set. The word occurrence pattern is revealed progressively as queries are issued. We could call this *progressive* L4 leakage.

**Appended-keywords SE.** Another class of SE schemes encrypts each document $D_i$ using conventional randomized symmetric encryption, and then append to the resulting ciphertext $C_i$ an encoding of the values $F_K(W_i[1])$, …, $F_K(W_i[c_i])$. Here $F_K$ is a pseudorandom function (PRF) such as HMAC. All of the ciphertexts resulting from performing this are uploaded as-is to the server. Search on a keyword $w$ is easy: compute $F_K(w)$ and request the server to perform a search on it.

As discussed at length in [HAJ+14, LCS+14], the benefit of appended-keyword SE schemes is that it is legacy-compatible: the server can perform indexing on the uploaded ciphertexts, addition and removal of keywords is straightforward, etc. One can also easily support legacy-compatible frequency counts or ranked keyword searches using information retrieval methods.

We note that in some implementations the client will sort $F_K(W_i[1]), \ldots, F_K(W_i[c_i])$ before uploading. This is better for security, and both ShadowCrypt and Memesis Aegis do so. If one does not sort on the PRF values, then information about the order of keywords found in the document is leaked by the order of PRF values. For example, two documents that have different orderings of their keywords will have different orderings of appended PRF values. We also consider unsorted appended-keywords SE because we believe it is not understood that this is less secure. For example, Lau et al. state that just outputting the PRF values in whatever order is secure [LCS+14, Appendix A] and apparently only used sorting for performance reasons by way of insertion of the PRF values into a Bloom filter.

If the keyword hashes are not sorted and repeats are included, (to allow searches informed by frequency information), then appended-keywords SE and substitution-cipher SE have identical leakage (when using the same keyword extraction algorithm).

Because of the requirements of legacy compatibility, these appended-SE schemes provide no additional hiding of occurrence patterns prior to search. Co-incidence relationships, counts of number of unique keywords, ciphertext length, and ordering (if sorts on the PRF values are not performed) are revealed immediately upon upload of the ciphertexts.

Thus, for this class of in-place schemes we define two leakage profiles, one for in-order and one for random-order encrypted keyword upload:

L3: *Fully-revealed occurrence pattern with keyword order.* Intuitively, this leakage profile fully reveals the pattern of keyword occurrences in documents, in the order of their first appearance, but not the occurrence counts within a document. Formally, using the notation from above, the profile outputs the sequence of sets

$$\{\{(i,j) : W_i[j] = w_1\}, \ldots, \{(i,j) : W_i[j] = w_N\}\}.$$

That is, the first set includes all pairs $(i, j)$ where $w_1$ appears first as the $j$th term in the processed version of document $i$. This leakage profile occurs in in-place schemes when hashes of the keywords are appended to uploaded documents in their original order of occurrence in the document. It is equivalent to releasing key-ed hashes of every document vector $W_i$, in order.

L2: *Fully-revealed occurrence pattern.* This profile is similar to leakage profile L3 in that the occurrence patterns of keywords are revealed for every term, yet not in document order. Formally, if the documents collectively contain terms $\bigcup_i W_i = \{w_1, \ldots, w_N\}$, then the profile leaks the full collection of sets

$$\{i : w_1 \in W_i\}, \ldots, \{i : w_N \in W_i\}.$$

This leakage is equivalent to the following: For each set $W_i$, compute keyed hashes of the terms, sort the hashes lexicographically, and then output the sorted set.

This profile arises in both in-place SE and the encrypted-index SE described in the next subsection. An in-place scheme will leak this information when hashes of each keyword are appended (in random order) to securely encrypted documents before uploading.

L2 is the leakage profile of the SE projects Shadowcrypt [HAJ$^+$14] and Mimesis Aegis [LCS$^+$14], both of which use an appended-keywords scheme.

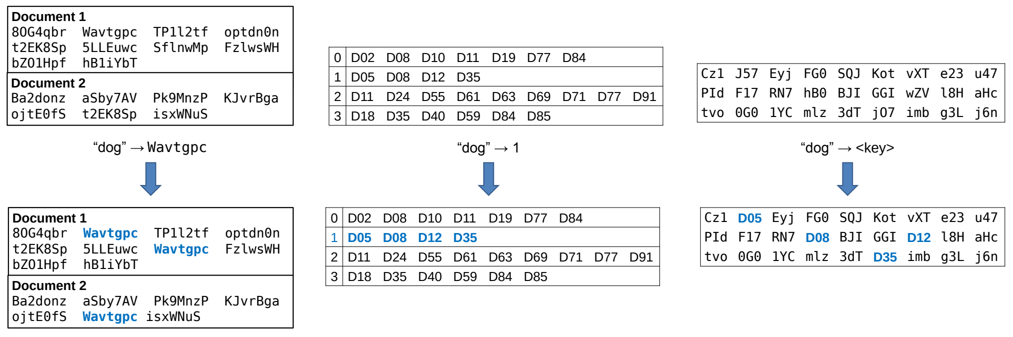

Document 1
80G4qbr Wavtgpc TP1l2tf optdn0n
t2EK8Sp 5LLEuwc SflnwMp FzlwsWH
bZO1Hpf hB1iYbT
Document 2
Ba2donz aSby7AV Pk9MnzP KJvrBga
ojtE0fS t2EK8Sp isxWNuS
"dog" → Wavtgpc
Document 1
80G4qbr Wavtgpc TP1l2tf optdn0n
t2EK8Sp 5LLEuwc Wavtgpc FzlwsWH
bZO1Hpf hB1iYbT
Document 2
Ba2donz aSby7AV Pk9MnzP KJvrBga
ojtE0fS Wavtgpc isxWNuS
0 D02 D08 D10 D11 D19 D77 D84
1 D05 D08 D12 D35
2 D11 D24 D55 D61 D63 D69 D71 D77 D91
3 D18 D35 D40 D59 D84 D85
"dog" → 1
0 D02 D08 D10 D11 D19 D77 D84
1 D05 D08 D12 D35
2 D11 D24 D55 D61 D63 D69 D71 D77 D91
3 D18 D35 D40 D59 D84 D85
Cz1 J57 Eyj FG0 SQJ Kot vXT e23 u47
PId F17 RN7 hB0 BJI GGI wZV l8H aHc
tvo 0G0 1YC mlz 3dT j07 imb g3L j6n
"dog" → <key>
Cz1 D05 Eyj FG0 SQJ Kot vXT e23 u47
PId F17 RN7 D08 BJI GGI D12 l8H aHc
tvo 0G0 1YC mlz 3dT D35 imb g3L j6n

Figure 5.3: Server's view SE data structures for the same plaintexts before and after searching for the keyword 'dog'. From left to right, the schemes are examples of leakage profiles L4, L2, L1.

To provide an intuitive sense of the leakage profiles, Figure 5.3 shows simplified diagrams of the data structures that the server sees, before and after the word "dog" is queried, for three schemes with leakage profiles L4, L2, and L1, respectively. The text to the right of the arrow represents the form in which the query is sent to the server. In the center (L2) scheme, the entries of the table (starting with D) are document identifiers.

#### 5.3.1.2 Encrypted-index SE

Most academic work has been on (what we call) encrypted-index SE [SWP00, CGKO06, KPR12, KP13, NPG14, OKKM13, SPS14, CJJ$^+$14, CJJ$^+$13, KO13, Kur14]. In these schemes, clients construct an encrypted index before upload.

To describe these schemes, we need to introduce additional notation. For these schemes, the client generates an *inverted index* $\mathbf{I}$. This is a $m \times n$ matrix $\mathbf{A}$ where entry $\mathbf{A}_{i,j} = 1$ iff document $D_j$ contains word $w_i$. All other entries are zero. The $m \times m$ *co-occurrence count matrix* $\mathbf{C}$ contains in location $\mathbf{C}_{i,j}$ the number of documents in which $w_i$ and $w_j$ both occur. This can be normalized to produce an empirical *co-occurrence probability matrix* $\hat{\mathbf{C}}$.

Searching the index **I** requires the client to generate a per-query *trapdoor*, which is sent to the server. The trapdoor can be thought of as allowing the server to decrypt only those document identifiers corresponding to the search term. In some cases, a special search protocol is performed with the trapdoor by the server on the encrypted index. For instance, the protocol of [CJJ$^+$13] employed an interactive protocol to carry out Boolean searching.

Since in index-based schemes, the ordered document texts are not sent to the server, encrypted or otherwise, but only the index **I** (possibly in an encrypted format), encrypted-index SE reveals neither the order of words in a document, nor the number of times a word occurs in a document.

**Unencrypted Inverted Index.** In this scheme, the inverted index **I** (table of document IDs) is sent unencrypted to the server. We assume the server cannot use the document ID's to access the document plaintexts. The client randomly permutes the rows of the table before uploading, perhaps according to a PRP, so the server does not know the correspondence between rows and keywords. To fulfill a search request, the server simply returns the appropriate row to the client.

Note that Boolean queries can be fulfilled at no additional security loss, since the server already sees the whole table.

Since this scheme is essentially giving the server the document-term matrix, the server can, prior to any queries being issued, directly observe the length of all result sets and also construct the co-occurrence matrix **C** for all keywords. In fact, this scheme has leakage profile L2, equivalent to the appended-hash scheme described above. Specifically, when the term-document index is uploaded without encryption, an adversary can derive the same information as provided by appended keyword hashes.

**Encrypted Inverted Index, no result length hiding.** This scheme differs from the above in that each row of the index, in the form of a list of document identifiers (i.e. nonzero matrix entries), is encrypted as a whole (say with a block cipher in some randomized chaining mode). In this way, no repeated document IDs can be read from the index before queries are issued. However, the length of each row is not hidden; before any queries are issued, the server can observe the result *count* for every row.

The client searches by sending a key allowing the server to decrypt the index row corresponding to the keyword. As queries are fulfilled, the server gains information about the overlap of documents in result sets.

In this scheme the server can fulfill Boolean queries by obtaining the decryption key for every keyword in the query. However, this results in the server learning additional document ID's beyond those satisfying the query.

We do not define a separate leakage profile for this scheme, but consider it under profile L1, described next.

**Fully Length-hiding SE.** This strongest class of searchable encryption schemes hide the length of the result sets before they are queried, so that before queries are issued the server learns nothing except the total size of the index.

It was suggested by Song et al. [SWP00] that the length of result lists could be partially hidden by padding the shorter lists to a fixed size. A solution to completely hide result lengths without using padding was first presented in Curtmola et al. [CGKO06], by the use of an interwoven linked list data structure. Recent work has shown that it is possible to fulfill Boolean queries in this setting while revealing less information than the entire row for all words in the query [CJJ+13].

We define the leakage of this fully-length-hiding SE, the smallest leakage profile we consider, as L1:

L1: *Query-revealed occurrence pattern.* Intuitively, this profile reveals the same information as L2, but only for terms that have been queried. This is the class of leakage targeted by [CGKO06] and its derivatives. In this profile, initially only basic size information (e.g. the total length of the documents) is leaked. When a term is queried, the profile includes the *access pattern* of the term, which means identifiers of documents containing $w$.

Formally, when $\mathbf{D}$ is processed into $\mathbf{W}$ and encrypted, and a sequence of queries $q_1, q_2, \ldots, q_Q$ is issued, then the leakage profile includes the sequence of sets

$$\{i : q_1 \in W_i\}, \{i : q_2 \in W_i\}, \ldots . \{i : q_Q \in W_i\}.$$

Depending on the setting, the leakage function may permute the indices $i$ randomly to hide the initial correspondence between the plaintexts and ciphertexts. This means for each queried term, the profile leaks the number of documents containing the term, and for multiple queries, the profile reveals which documents they have (or do not have) in common.

For a simplified illustration of what the server sees before and after a query in schemes with leakage profiles L4, L2, and L1, see Figure 5.3.

### 5.3.2 Attack Models

We classify attack models along two axes. First, we consider the *mode* of the attack, meaning whether the server is passively or actively mounting attacks, and second, we specify the *knowledge* of the server regarding the documents and queries.

**Attack modes.** In SE the server receives the ciphertexts and query requests from a client, and an adversarial server may use its position to extract private information. We classify the following three attack modes (where the last two may be utilized simultaneously in one attack). In each case, the adversary is a server following the SE protocols, but it may take actions to induce the client to run with certain inputs.

- An *honest-but-curious* server follows the protocol and takes no actions beyond those of an honest server, and attempts to learn about the plaintext documents of the terms that were queried.
- An active adversary can could carry out a *chosen-document attack* where it tricks the client into include a chosen document in the document set. Our model scenario for this attack is an automated-update email or document storage system, where new documents are processed and added without explicit human involvement. In this setting the adversary will be able to craft a document, induce the client into including it, and then learn from the subsequent leakage (along with knowledge of the chosen document, of course).

An adversarial server may mount a *chosen-query attack* by inducing the client into issuing certain queries, thereby revealing whatever is leaked by that query. Our attacks will not use attack mode as it seems less realistic, but it is conceivable that a deployment scenario would enable this ability in some form.

**Adversary knowledge.** One of our theses is that an SE server's prior knowledge of the documents and queries may enable the extraction of more information. We specify the following possible types of prior knowledge. Depending on the setting, multiple types of knowledge may be available to the adversary.

- *Distributional query knowledge* models a typical case where an adversary has some idea about the queries being issued. For instance, if encrypted chat logs are being searched, an adversary can refer to typical user behavior regarding those searches to inform its attack.
- *Known queries* occur when the server knows some of the terms input by the client for a search. In practice an adversarial server may use contextual information about the client's behavior to accurately guess some queries. Formally, in this setting we will draw some queries $q'_1, \dots, q'_k$ from a distribution that all become known to the adversary, and then other queries $q_1, \dots, q_Q$ from a distribution that the adversary will not know.
- *Distributional document knowledge* models the contextual information an adversary will have about the documents (e.g. whether they are emails, or corporate sales documents, or something else). Formally we will model this by considering a distribution on documents. In our attacks, we will divide large data sets into two parts, giving one part to the adversary.
- *Known documents* model scenarios where an attacker will know certain plaintext documents or perhaps have significant information about them. For example, an attack might know that a widely-distributed email is somewhere in a user's repository. In our attacks we model this situation by drawing documents as above, and then either hand-selecting likely known documents or choosing some at random.
- *Fully-known document set* is the setting explored by IKK [IKK12], where all of the documents are known to the adversary, and only some or all of the queries are unknown.

**Attack objectives.** We next identify possible objectives for an adversary controlling the server in an SE scheme.

- *Query recovery (QR)* is the goal of determining the plaintext of queries that have been issued by the client. This was the objective of the first known attack on SE by Islam et al. [IKK12]. QR is non-trivial in any setting where some of the queries are known, including cases where the documents are fully or partially known.

- *Plaintext recovery (PR).* In PR attacks, the adversary's goal is to reconstruct as much as possible of the client's indexed documents, primarily by learning a mapping of keywords to their encrypted versions (informally, hashes.) A PR attack may reveal plaintexts as a "bag-of-words" or in document order; for the attacks we present, this depends on the scheme and is independent of the attack method itself.

Of course, these are not the only attack objectives: we briefly point out two others, *document presence* and *document identification*. In a document presence attack, the adversary simply wishes to determine whether a known plaintext document is present in the client's index. In a document identification attack, the server seeks to find the correspondence of known documents to the document IDs revealed by the SE scheme. Also note that attack objectives are interrelated. In particular, a successful plaintext recovery attack will make document identification trivial.

In the subsequent sections we describe our attack results for meaningful combinations of attack objective, leakage profile, and server knowledge. See Figure 5.1 on page 70 for a summary.

## 5.4 Experimental Methodology

We investigated the vulnerability of the described leakage profiles by means of simulated query recovery and plaintext recovery attacks, using two separate publicly available email datasets. The first is emails from 150 employees of the Enron corporation from 2000-2002, available online [Enr]. In order to focus on intra-company email, following the approach of [IKK12], we took emails from each employee's sent mail folder, resulting in 30,109 total documents. The on-disk size of the data set is 50 megabytes. The second dataset we used is a subset of the Apache mailing list archives. Specifically, we used the "java-user" mailing list from the lucene project for the years 2005-2011. This consists of 50,582 emails, with an on-disk size of 338 megabytes.

For each dataset, one email message is considered as one document. A fixed-size vocabulary was established by taking the top keywords from each dataset after removal of 200 stopwords. In experiments where the vocabulary size was not varied, we used a keyword set of size 5000. This gives an average of 93 words per document in the Enron corpus, and 291 in the Apache dataset. Keywords are stemmed using the standard Porter stemming algorithm. Stemming is a crucial feature of usable search functionality, as it provides more flexible matching, e.g., a search for “cat” will also match the word “cats”. Stemming has a two-sided effect on the attacks that we evaluated. Firstly, it allows the adversary to reconstruct only the stems of keywords, so number and tense information of the original plaintext cannot be recovered. On the other hand, stemming reduces the total vocabulary size and increases repetitions of terms, making the attack easier.

## 5.5 Query Recovery Attacks

These attacks apply to any of the leakage profiles defined above (L1 or greater), being originally designed for the indexing-based full SE schemes described in Section 5.3.1.2. Again, the adversary’s goal in query recovery attacks is to recover the correct plaintext keywords corresponding to queries that the client has issued to the SE server.

These attacks depend on multiple queries being issued, correlating the access pattern leakage. These attacks work under the lowest leakage profile, L1 (the leakage of full SE), but require the server to have more extensive knowledge of the document set that is indexed.

### 5.5.1 Prior Work: The IKK Attack

Islam et al. [IKK12] give the first successful experimental attack on searchable encryption that we are aware of. Theirs is a query recovery attack on SE (leakage profile L1), using full document knowledge and partial query knowledge. They give experimental results using the Enron email dataset, achieving recovery rates of up to 80% of issued queries.

The attack presumes a fixed set of $m$ potential search terms, and the server’s knowledge of $\mathbf{D}$ is distributional, in the form of the $m \times m$ matrix $\hat{\mathbf{C}}$ of word co-occurrence probabilities. Note that this can be computed exactly if the server knows the true document-term matrix $\mathbf{A}$ for the indexed documents.

The server mounts the attack by observing the document overlap pattern revealed by the client's issued queries. Let $q$ be the number of unique query tokens observed. These are used to construct a $q \times q$ term co-occurrence matrix $\mathbf{C}_t$ and its normalized version $\hat{\mathbf{C}}_t$, which is a permutation of an unknown submatrix of $\hat{\mathbf{C}}$ (approximate if the server's knowledge is not exact.) Then, simulated annealing is used to find the best match of $\hat{\mathbf{C}}_t$ to $\hat{\mathbf{C}}$. The output is a mapping of rows of $\mathbf{C}_t$ to rows of $\hat{\mathbf{C}}$, which are the guesses for the query terms. In leakage profile L2 or greater, the attack can be carried out using the entire keyword set prior to observing any queries.

The authors give experimental results for the Enron email dataset. The keywords are taken to be the $m$ most common stemmed words in the document set after stopword removal, and the queries are chosen uniformly at random from among these. The recovery rate reported varies from near perfect with 500 keywords to approximately 0.65% with 2500 keywords. Holding the number of unique queries $t$ constant at 150. We re-implemented the attack to confirm these results and make further comparisons.

One strength of the attack, henceforth called the IKK attack, is that the success rate is largely independent of the number of queries issued. The median number in the results of [IKK12] is 150. Thus, in the remainder of this work we use either 150 queries or 10% of the keywords as a default, whichever is larger.

A significant weakness of the attack is that the adversary's advantage does depend strongly on the number $m$ of keywords under consideration. The authors of [IKK12] do not report on experiments with keyword sets larger than 2500. In our experiments, the annealing attack performs poorly on queries for vocabulary sizes over 5000. This is shown in Figure 5.4. Thus the attack of [IKK12], while valuable for illustrating the vulnerabilities of SE, is fairly non-scalable.

As will be shown in Section 5.5.3, while the IKK attack explicitly requires only distributional server knowledge, in practice it requires the adversarial server to have highly accurate co-occurrence knowledge of the true document set. This led us to consider a more direct attack that can be carried out when the server does have explicit access to the true document set.

### 5.5.2 Query Recovery with a Counting Attack

When the adversarial server, in addition to knowing the co-occurrence pattern of keywords, also knows the number of documents in the indexed set that match each keyword—the *result lengths*—a much more efficient and accurate query recovery attack is applicable. This length-based query recovery attack is applicable to all schemes with leakage profile L1. The attack is efficient and requires no numerical optimization techniques.

**Attack Description.** Similarly to the IKK attack, the adversarial server uses keyword co-occurrence matrix $\mathbf{C}$, and a fixed number $m$ of keywords is considered. In addition, we assume that the server also knows or can compute the number of matching results $\mathbf{count}(w)$ for each keyword $w$ in the true document set. Note that all this information can be easily computed if the server has access to the explicit document set.

The first observation is that if any keyword with a unique result count is queried using trapdoor $q$, then a server with knowledge of the true document set can immediately recover the query, by finding the word $w$ such that $\mathbf{count}(w) = \mathbf{count}(q)$.

In the Enron email dataset with the same setup as [IKK12], if we consider the most common 500 non-stopwords as keywords, 63% of them have a unique result count. If 2000 keywords are considered, then 24% are unique. Note that due to the Zipfian character of word distribution in natural-language text, the unique result counts will come from the more frequently occurring keywords. If queries are assumed to be drawn uniformly from this set of keywords, then these percentages can be thought of as a baseline rate for query recovery when the server knows result counts. Also note that the larger the document collection, the larger the number of keywords that have a unique result count (although the total number of keywords present grows as well.)

Beginning with this baseline knowledge, our attack algorithm proceeds to recover queries that do not have a unique result count by comparing term co-occurrence counts. See Algorithm 1.

In contrast to the IKK attack, in the count attack the adversary definitely knows which queries have been correctly reconstructed, and has partial information about others in the form of the candidate set.

```
Input: Unencrypted keyword index Index, observed query tokens $\mathbf{t}$ and results
Initialize known query map $K$ with queries $(q, k)$ having unique result lengths;
Compute co-occurrence counts $C_q$ for observed queries and $C_I$ for Index;
while size of $K$ is increasing do
    for each unknown query $q \in \mathbf{t} - K$ do
        Set candidate keywords $S \subseteq K = \{s : \mathbf{count}(s) = \mathbf{count}(q)\}$;
        for $s \in S$ do
            for known queries $(q', k) \in K$ do
                if $C_q[q, q'] \neq C_I[s, k]$ then
                    remove $s$ from $S$;
                end
            end
        end
        if one word $s$ remains in $S$ then
            add $(q, s)$ to $K$;
        end
    end
end
```

**Algorithm 1:** The count attack algorithm.

**Count Attack Analysis.** Figure 5.4 shows the query reconstruction results from our implementation of the count attack in the same setup as [IKK12], but with the server having no initial knowledge of any queries. The graph shows accuracy results for varying number of keywords from 2500 to 15000, in each case assuming that 10% of the keywords under consideration have been queried. The attack achieved perfect reconstruction for keyword counts less than 2500. Since the count attack takes only a few seconds to run, compared with hours for the simulated annealing, we could run it with much larger numbers of keywords than IKK [IKK12].

Our experiments indicate that once even a small number of queries is initially disambiguated, the co-occurrence counting phase can successfully recover nearly all the queries.

**Padding Countermeasures.** We considered whether the effectiveness of the count attack could be decreased by a client who pads the index with additional entries for bogus document ID's. These bogus ID's can be filtered from search results by the client.

We consider a scheme in which the number of entries in each index row is padded up to the nearest multiple of an integer $n$. This can be thought of as constructing

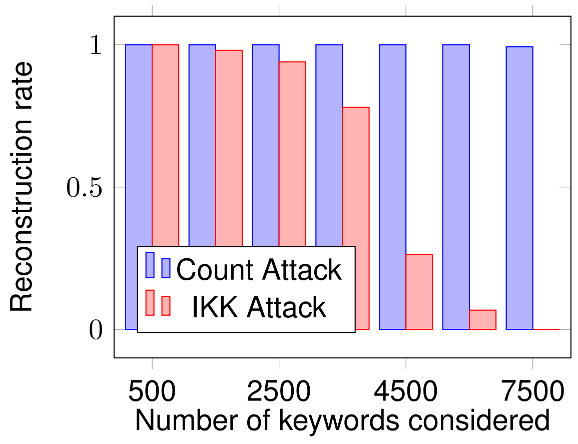


Figure 5.4: Comparison of IKK and Count attacks for query reconstruction by vocabulary size. Enron dataset, 150 keywords queried.

query "buckets" within which keywords will have the same result length. Changing the padding size $n$ allows us to adjust the space-security tradeoff. To avoid statistical analysis to determine which document IDs are used as padding, a bogus document set can be constructed by sampling from the same term distribution as the true documents. Then the desired padding entries can be chosen from among these.

Qualitatively, this affects the count attack in two ways: Firstly, it reduces the number of unique result lengths, increasing the number of candidate matches for a given query. Secondly, it prevents the co-occurrence counting stage of the count attack from being carried out exactly: The number of co-occurrences of two keywords in the padded index may exceed the co-occurrence count of the corresponding keywords computed from the true document set. The attacker must allow for this and cannot eliminate as many candidates.

The count attack as described above completely fails once the padding size $n$ is increased to the point that no keyword has a unique result count; in this case it cannot obtain even a single initial known query with which to bootstrap the co-occurrence counting. However, we can modify the algorithm to allow the attack to be carried out with reduced information. There are two main insights in the *generalized count attack*. Firstly, since the server is aware that the padding can cause additional "false co-occurrences", it can adjust its comparison so that, instead of requiring an exact match of co-occurrence counts, it accepts co-occurrence counts within a window as large as the maximum number of false co-occurrences. Secondly, we remove the algorithm's dependence on initially finding a query with a unique result count. This is done by finding the result length among the queries that has the smallest number of occurrences, provisionally setting the query to the first one, and running the remainder of the algorithm. If the guess is wrong, the co-occurrence counting phase detects an inconsistency and bails out, and the next candidate will be tried.

These modifications to the count attack algorithm maintain its correctness. Figure 5.5 shows the effect of padding on the generalized count attack.

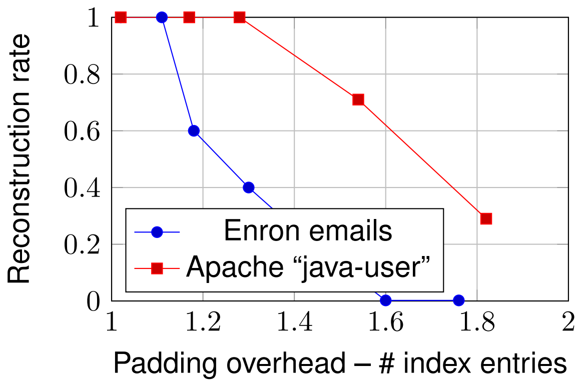


Figure 5.5: Generalized count attack results for padded index, 5000 keywords considered, 10% of keywords queried.

Up to a padding level that increases the index size by about 15% for the Enron data, and 30% for the Apache data, the attack's success rate is unaffected. We believe the attack is more robust on the Apache email because the dataset is more dense—each document has more words, so there is more co-occurrence information. The recovery rate drops off rapidly after this, as the data becomes sufficiently noisy to prevent any disambiguation. Note that unlike with the IKK attack, these results are with the server knowing zero queries initially.

### 5.5.3 Query Recovery from Partially Known Documents

In this section we analyze the above attacks on L1 leakage (with slight modifications) in the case when the server has only partial knowledge of the document set, to better understand the server knowledge level required to carry out these attacks.

**Analysis of the IKK attack with partial knowledge.** Because the IKK attack only directly makes use of word co-occurrence probabilities, technically it does not require the server to have explicit knowledge of the documents themselves. However, the effectiveness of the attack depends strongly on the accuracy of the co-occurrence matrix. The authors discuss this and give results of an experiment in which random Gaussian noise is added to the co-occurrence matrix. In their experiment, the recovery rate drops from 85% to 65% at the first increment of added noise.

To understand better how real-world limitations on a server's knowledge affects the accuracy of the attack, we devised a new experiment. Instead of adding noise to the co-occurrence matrix, we assume the server knows only a certain *fraction* of the true documents, and compute the co-occurrence probabilities from that. We duplicated the experiments of [IKK12] with the server's co-occurrence matrix constructed from various percentages of the true document set. See results in Figure 5.6, which relate the recovery rate to the percentage of the dataset known to the server. In brief, these results indicate that unless the server has access to 99% of the true document data, the query recovery rate is quite poor.

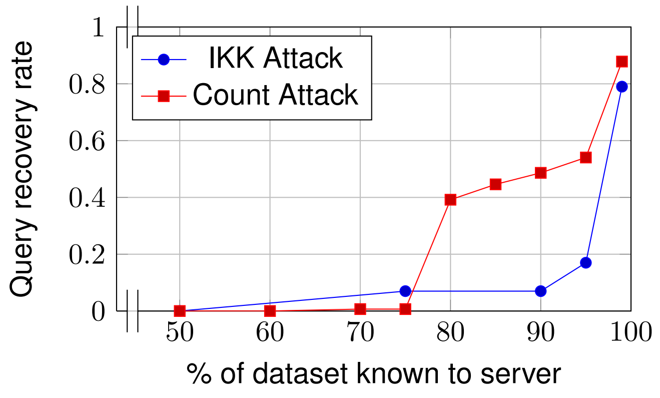


Figure 5.6: Query recovery rates when server has partial knowledge of true document set. Enron dataset, 500 keywords, 150 queried uniformly. Server knows 5% of queries at start.

Note that this is the case what the server knows is a subset of the true documents. We also conducted experiments showing that if the server is trained on a random subset of documents, while the client queries a disjoint subset from the same corpus, (the unknown documents case), the IKK attack fails completely.

**Generalized count attack for partial knowledge.** We also modified the generalized count attack described in the previous section to allow query reconstruction when the server does not know the full document set. As with the case of padding, this requires the algorithm to allow keyword candidates within a window of co-occurrence counts, rather than requiring exact equality. These results are also plotted in Figure 5.6. Note that the count attack performs better, requiring the server to know only 80% of the dataset for significant query recovery.

These data support the conclusion that attack on L1 leakage require a significant amount of server knowledge of the document set, but are nonetheless possible with less-than-perfect knowledge.

## 5.6 Document Identification Attacks

The same type of result length and term co-occurrence information can be used to carry out a document identification attack as well. Recall that document identification is the task of determining the correspondence between document IDs observed by the server and a set of documents known to the server.

In terms of the term-document matrix $\mathbf{A}$, Document Identification can be seen as the "dual" task of Query Recovery—the goal is to recover the correct column permutation. However, the count attack as described above cannot be applied directly on the transposes of the matrices (in the case of full SE). In that case, the result counts desired would be the number of terms in a given document. However, it is still terms that are being queried, and rows, not columns, of $\mathbf{A}$ are revealed to the server. As long as only a subset of terms is queried, the server does not learn the exact count of terms in any document.

Given this, the most straightforward document identification attack in this setting may be to carry out the query recovery count attack, and then use that information to recover the correspondence of document IDs to known documents.

**Attack algorithm.** Assume the count attack above has been successfully carried out, and that the server now holds a query matrix $\mathbf{B}$ for which it knows the correspondence of rows to queries. Then $\mathbf{B}$ is a known subset of the rows of $\mathbf{A}$, with the columns randomly permuted (from the server's point of view.) The algorithm is simple. Each column of $\mathbf{B}$ (document) is examined one-by-one and compared to the server's documents to see if the term occurrences match. This produces a set of candidate documents for each column; if the size of the candidate set is one, the document has been uniquely identified. As with the query recovery count attack, multiple passes can be used to eliminate additional candidates.

**Document Identification Using Payload Lengths.**

Searchable encryption schemes also become vulnerable to attack in the case where the server has access to the sizes of the document payloads. This happens whenever the server used for searching is the same server that retrieves the documents, or the adversary controls both. Payload length information can be used to re-identify keywords and documents in the same manner as the above attacks that use result count information.

## 5.7 Plaintext Recovery Attacks

In plaintext recovery attacks, the server's goal is to learn the mapping of keywords to the ciphertexts stored by the SE scheme, for as many keywords as possible. This in turn allows reconstruction of stored documents, either as a "bag of words" in appended-keywords schemes, or ordered plaintext if the document is deterministically encrypted using the word hashes.

These attacks exploit leakage profiles L2 and L3. For example, they apply to searchable encryption schemes that store encrypted words on a per-document basis using a PRF or hash function, as in the "in-place" schemes as described in Section 5.3.1.1. We show realistic and highly damaging known-document (passive) and chosen-document (active) attacks in this scenario. The server is required only to know a small number of stored documents, as described in each experiment.

### 5.7.1 Passive Attacks

This section presents and gives analyses of attacks for query recovery and keyword-based plaintext recovery by a passive server that correctly executes the SE scheme algorithms. Attacks are classified by the SE scheme type (substitution or indexing-based), and how much prior knowledge of the plaintext document set **D** the server possesses.

#### 5.7.1.1 Order of Hashes Known (L3)

To start with a simple case, we consider a scheme in which the order of appended hashed keywords is not changed from the order in which the keywords appear in the document. This is leakage profile L3. In this case, all the indexed keywords in any known document are immediately revealed to the server. We present the results of statistical experiments quantifying the advantage gained by an attacking server in this scenario.

**Random Documents.** To determine the fraction of plaintext keywords learned by an adversarial server that knows a small number of the stored documents, we computed the fraction of documents at a given recovery rate, for varying number of known document and 20 known random documents, averaged over 10 random trials in each case. Results for the two datasets, for 2 and 20 known documents, are summarized in cumulative style in Figure 5.7. The curves which fall further to the right are indicative of a larger percentage of documents having high keyword recovery rates. As mentioned, the datasets have over 30,000 and 50,000 documents, respectively, so even when a very small fraction of the documents is known to the server, the server can see a substantial percentage of the words of the stored documents.

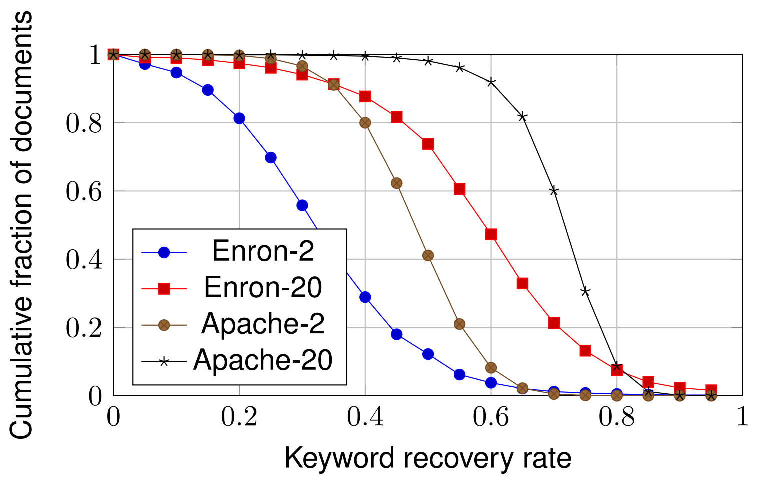


Figure 5.7: Plaintext reconstruction rates for the Enron dataset from known documents for an in-place SE scheme with ordered keyword hashes.

These results show that even a small number of known documents allows the server to recover a significant percentage of the documents, enough that a human inspecting the output in the form of redacted documents may obtain a very strong sense of its

```
 The attached contract is ready for signature.  Please print
2 documents and have Atmos execute both and return same to my
attention.  I will return an original for their records after ENA
has signed.  Or if you prefer, please provide me with the name /
phone # / address of your customer and I will Fed X the Agreement.
```

```
attach contract signatur pleas print 2 document have execut both same
will origin ena sign prefer provid name agreement
```

Figure 5.8: **(Top)** An example plaintext email from the Enron corpus. **(Bottom)** The stem words recovered by our attack when given 20 randomly selected known emails.

content.

Note that the curve is steeper for the Apache dataset results. Our hypothesis is that the Apache dataset has a "critical mass" of vocabulary that is common to most of the documents, as the topic of discussion in a software mailing list is likely to be more uniform than all the emails sent by a large company's employees.

To get a sense for the ability of a human attacker to gain information from this type of reconstruction, we took the stemmed keywords from a 20-document Enron trial, made a random selection of several other Enron emails, and printed out the first occurrence of each matching stem in document order, omitting stopwords. A sample is shown in Figure 5.8. Note that potentially sensitive information has been revealed, including the name of a company involved in a contract.

**Known Public Documents.** Though choosing documents at random is important for statistically understanding the power of attacks, the true source of a known-document attack would likely not be an email chosen uniformly at random. A more probable source might be a message that has a wide distribution, such as a company-wide announcement. The more recipients an email has, the more likely it is that its plaintext will become available to an attacker.

To test this, we ran the same experiment with a *single* email from the Enron dataset that was sent to 500 recipients. It was an announcement sent an entire division, four paragraphs long, with 832 unique keywords, containing an announcement of an upcoming survey of the organization by an outside consulting group. From this single document, an average of 35% of the indexed keywords in every document could be recovered.

#### 5.7.1.2 Order of Hashes Unknown (L2)

In leakage profile L2, for example an in-place scheme when the keyword hashes are stored in randomized order, a server who knows a number of document plaintexts cannot immediately determine the PRF mapping of every keyword in the document, though he knows the *set* of hashes that correspond to the keywords in the document.

We can quantify the ambiguity of words in the known documents. A keyword which the server knows the hash of has ambiguity 1 (no ambiguity). If the number of indexed keywords in a document known to the server is $w_1$, then for each word there are $w_1$ different possibilities for its hash. If this is the only document known to the server, we say the average ambiguity of all the keywords in the known document set is $w_1$.

When more than one document is known by the server, the ambiguity can be reduced, again by use of the co-occurrence pattern of keywords in the known documents. We omit a full analysis of this approach. A much more powerful attack for unknown order of hashes is possible when the server is able to plant documents, as shown in section 5.7.2.2.

### 5.7.2 Active Attacks

We now consider the power of an attacking server who, in addition to observing the client's uploaded data, can "plant" documents that will be processed by the client and added to the uploaded dataset. Note that the protocol itself is not attacked; the server still fulfills requests following the rules of the scheme. This attack model can be seen as the analogue of the chosen-plaintext attack model (CPA) used in security proofs for encryption schemes.

We can easily imagine real-world scenarios where attacks of this nature can be carried out—it can be as simple as a malicious server sending a client an email, which is then indexed by the client and uploaded to the server. As another example, we described how a malicious server might make an invisible-to-the-user attack on ShadowCrypt using UI redressing in Section 5.2.

The attacks described below apply to in-place schemes, as described in section 5.3.1.1. We assume that by observing the payload length and/or the time at which documents are uploaded, the server can determine which index or database entry corresponds to the document he planted. For each attack we measure the level of plaintext recovery as percentage of keywords reconstructed.

#### 5.7.2.1 Hash order known (L3), chosen document

Recall that searchable encryption schemes with leakage profile L3 preserve the document order of encrypted keywords stored at the server, either by deterministic word-based encryption of the document itself or by storing keyword hashes in their order of appearance. In this case then an adversarial server can carry out a simple but devastating attack. He can plant a single document in the database with any desired set of keywords, and then from its encryption can learn all of the hashes of those keywords. We do not explore this simplistic, though obviously very damaging attack, further.

#### 5.7.2.2 Hash order unknown (L2), chosen documents

We now consider schemes in which the hashes of keywords are stored at the server in random order. In this case, an adversary who plants a chosen document will only learn the *set* of keyword hashes for the document, and not the one-to-one mapping of keywords to hashes. Clearly, the server can learn a single word unambiguously by planting a one-word document. An adversary would seek to maximize the "yield" of keywords learned per inserted document, minimizing the number of errors, and minimizing the number of inserted documents which in some settings could mean less chance of detection.

We present and analyze an attack based on frequency analysis of a related corpus, which allows the server to trade off error probability versus size and number of inserted documents, giving a highly effective attack strategy.

**Attack description.** From the known related corpus, the adversarial server generates a list of (possibly stemmed) keywords ranked by frequency. Fixing a document size $k$, the adversary divides the ranked keyword list into $k$ equal-sized slices. He then generates a $k$-word document by choosing the top word from each slice. The goal is to maximize the frequency distance between keywords in a given document.

The adversary also computes the frequency distribution of keyword hashes in the data uploaded by the client. After observing the hashes of the words in the planted document, the server ranks them by their frequency in the uploaded dataset, and guesses that the hashes correspond to the keywords of the same rank based on his own corpus. All $k$ guesses are correct as long as there are no rank reversals in their keyword frequencies between the two datasets. The adversary can repeat this process for all known keywords or for as many documents as he is able to insert.

We again use a simulated attack to estimate the keyword recovery rate and error rate of an adversary using this method.

**Experimental setup.** We used two setups, the first one to model an adversarial server that has access to a closely related corpus, the second to model a server having access only to unrelated text in the same language. For the first, we divided a single data set in half, with the server "training" on 50% of the documents to learn keyword frequencies, while the client processes the documents in the other half and uploads encrypted keywords to the server. For the second, we used keyword sets and frequencies from the Enron dataset to attack the Apache dataset, and vice versa.

In each experiment, we generated chosen documents to cover all the keywords known to the server and measured the average recovery rate as well as error rate of client keywords. The recovery rate takes into account errors as well as the fact that not all of the client-indexed keywords occur in the server's dataset, for which the server will never make a guess. The error rate counts only wrong guesses.

The plots in Figure 5.9 show the tradeoff between number of words per chosen document and error rate. The data are averaged over 10 runs per dataset and the two datasets. As expected, the larger the number of slices/keywords per document $k$, the greater the probability of error, as the words in a document will be closer in frequency, increasing the likelihood of a rank reversal between the client and server datasets.

In both the single-dataset 50% split and unrelated dataset scenarios, the error rate approaches the recovery rate as the number of terms per planted document increases. At the point where the error rate becomes higher than the attacker can be considered to gain very little information about keyword hash mappings. Also as expected, for the unrelated-corpus scenario, the recovery rate starts much lower, and a smaller slice size must be used to avoid too many errors. However, the error rate is not worse in the cross-dataset experiment, showing that frequency rank in English text is fairly

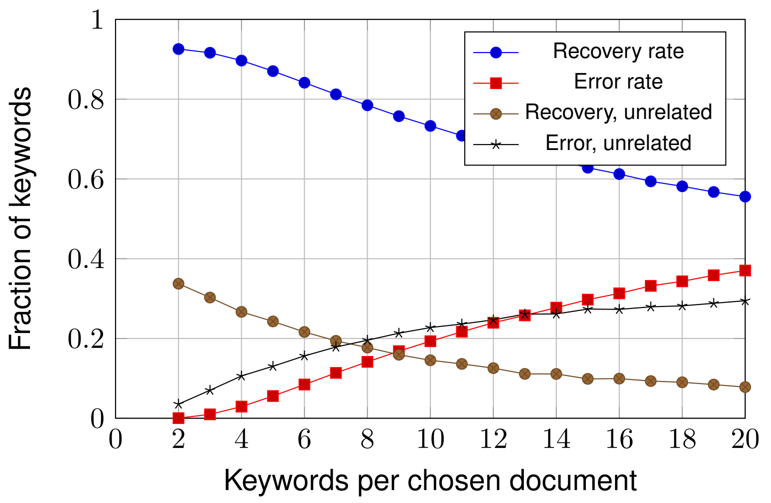


Figure 5.9: Keyword recovery rate and error rate for planted documents, unknown order of keyword hashes, 5000 keywords considered.

consistent, at least across these two email domains.

To recover $w$ keywords, the adversary must plant at least $w/k$ documents, more to allow for the error rate. The choice of document size must be informed by the attacker's desired error rate and the probability of detection. At any rate, the attack allows even an adversary who can plant a small number of documents to learn a sensitive selection of keywords.

## 5.8 Conclusion

These experiments, along with the classification of searchable encryption schemes, show that a malicious server with just a small ability to carry out active chosen-document attacks can expose a damaging quantity of private information from a broad class of searchable encryption schemes. If the server has only passive capabilities, equally damaging attacks can be carried out if the server has enough distributional knowledge of the document set.

Padding is an obvious defense technique for index-based schemes; however, we have seen that the information hiding accomplished by a padding technique needs to be sufficiently well understood to guarantee its effectiveness. We suspect that padding will be less useful against schemes that use word substitution, as the server has direct access to distributional knowledge of terms, and could potentially use passive distributional analysis to distinguish padding terms from legitimate keyword hashes. Investigating this could be a promising topic of future work.

# Chapter 6

# Insights to Drive the Deployment of Secure Computation

The work presented in this dissertation sheds light on key concerns in the use of secure computation, including usability of technologies based on provable security, efficiency when large datasets are involved, and the exploitability of leakage.

From Chapter 3, we learn that for general secure computation protocols to be gainfully deployed, it is necessary to not only translate protocols in the literature into implemented protocols, but also translate the *body of knowledge* in this literature to a representation that can be used to inform decision-making about the application of MPC protocols.

The work in Chapter 4 is an example of how scalability and usability concerns will continue to drive the development of application-specific protocols that may have leakage. It is simply not worth the trouble at the present time to model an entire database system in MPC. Nonetheless, the idea of looking for improved solutions by making use a semi-trusted third party was very much informed by general MPC results. Distributing trust among additional parties is a broadly applicable meta-technique for improving the security of computation without sacrificing efficiency, while retaining or even strengthening security.

Lastly, Chapter 5 sounds the alarm for a new kind of diligence that is needed when protocols with leakage begin to be sold and used in the world at large. Leakage, formally characterized or otherwise, will continue to be a reality. So we need to keep finding better ways to analyze what damage can be done with leakage.

Returning to the question asked in the introduction—"General or specific protocols?"—the work here has shown that the issue involves much more than a one-dimensional tradeoff between efficiency and security. It appears that, at least in the immediate future, more efficient protocols for specific secure computation tasks will continue to be adopted more widely than general MPC protocols. At the same time, as it has from the beginning, insights from the results in general multi-party computation will continue to be useful for driving the development of application-specific protocols. General MPC does, in fact, appear to be finally finding its niche, not in consumer applications but in the corporate and governmental realms, where the setup costs are more reasonable in proportion to the economic or political significance of a single computation.

# Bibliography

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography – TCC 2007*, pages 137–156, 2007.

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343. Springer Berlin Heidelberg, 2009.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC '93)*, pages 52–61, 1993.

[BCH12] Nir Bitansky, Ran Canetti, and Shai Halevi. Leakage-tolerant interactive protocols. In *Theory of Cryptography – TCC 2012*, pages 266–284, 2012.

[BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 186–195. IEEE, 2004.

[BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security*, pages 257–266, 2008.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. Cryptology ePrint Archive, Report 2014/075, 2014. `http://eprint.iacr.org/`.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *Advances in Cryptology – EUROCRYPT 2011*, pages 169–188, 2011.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, pages 479–488, 1996.

[Bea97] Donald Beaver. Commodity-based cryptography (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 446–455, 1997.

[Bea98] Donald Beaver. Adaptively secure oblivious transfer. In *Advances in Cryptology—ASIACRYPT '98*, pages 300–314, 1998.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88)*, pages 1–10, 1988.

[BKLPV13] Dan Bogdanov, Liina Kamm, Sven Laur, and Pille Pruulmann-Vengerfeldt. Secure multi-party data analysis: end user validation and practical experiments. Cryptology ePrint Archive, Report 2013/826, 2013. `http://eprint.iacr.org/2013/826`.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, pages 192–206, 2008.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC '90)*, pages 503–513, 1990.

[Bog13] Dan Bogdanov. *Sharemind: programmable secure computations with practical applications.* PhD thesis, University of Tartu, 2013.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145, 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88)*, pages 11–19, 1988.

[CD05] Ronald Cramer and Ivan Damgård. Multiparty computation, an introduction. In *Contemporary cryptology*, pages 41–87. Springer, 2005.

[CDdV$^+$05] Alberto Ceselli, Ernesto Damiani, Sabrina De Capitani di Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Modeling and assessing inference exposure in encrypted databases. *ACM Trans. Inf. Syst. Secur.*, 8(1):119–152, 2005.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography – TCC 2005*, pages 342–362, 2005.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology – EUROCRYPT 2000*, pages 316–334, 2000.

[CDN13] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing: An Information Theoretic Approach.* Self-published manuscript, 2013. `https://users-cs.au.dk/jbn/mpc-book.pdf`.

[CGKO06] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 79–88, 2006.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1985)*, pages 383–395, 1985.

[CJJ$^+$13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 353–373, 2013.

[CJJ$^+$14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*, San Diego, California, USA, February 23–26, 2014. The Internet Society.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *Advances in Cryptology – EUROCRYPT 2003*, pages 68–86. Springer, 2003.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing (STOC '86)*, pages 364–369, 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 494–503, 2002.

[CO15] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. Cryptology ePrint Archive, Report 2015/267, 2015. `http://eprint.iacr.org/`.

[DCCMP14] Giovanni Di Crescenzo, Debra Cook, Allen McIntosh, and Euthimios Panagos. Practical private information retrieval from a time-varying, multi-attribute, and multiple-occurrence database. In *Data and Applications Security and Privacy XXVIII*, pages 339–355. Springer, 2014.

[DCFG$^+$14] Giovanni Di Crescenzo, Joan Feigenbaum, Debayan Gupta, Euthimios Panagos, Jason Perry, and Rebecca N. Wright. Practical and privacy-preserving policy compliance for outsourced data. In Rainer Bhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *Financial Cryptography and Data Security*, volume 8438 of *Lecture Notes in Computer Science*, pages 181–194. Springer Berlin Heidelberg, 2014.

[DGKN09] Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *Public Key Cryptography*, pages 160–179, 2009.

[DKL$^+$13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority – or: Breaking the SPDZ limits. In *18th European Symposium on Research in Computer Security (ESORICS 2013)*, pages 1–18, 2013.

[DO10] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *Advances in Cryptology – CRYPTO 2010*, pages 558–576, 2010.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, pages 643–662, 2012.

[dVFJ$^+$07] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Over-encryption: Management of access control evolution on outsourced data. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 123–134, 2007.

[EG07] Sergei Evdokimov and Oliver Günther. Encryption techniques for secure database outsourcing. In *Computer Security - ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, 2007, Proceedings*, pages 327–342, 2007.

[Enr] Enron email dataset. `https://www.cs.cmu.edu/~./enron/`. Accessed: 2015-05-13.

[FIM$^+$01] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. In *ICALP*, pages 927–938, 2001.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88)*, pages 148–161, 1988.

[FPRSJ04] Joan Feigenbaum, Benny Pinkas, Raphael Ryger, and Felipe Saint-Jean. Secure computation of surveys. In *EU Workshop on Secure Multiparty Protocols*. Citeseer, 2004.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178, 2009.

[GIKR02] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 178–193. Springer Berlin Heidelberg, 2002.

[GKK$^+$12] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *ACM Conference on Computer and Communications Security (ACM CCS 2012)*, pages 513–524, 2012.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 555–564, 2013.

[GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *Distributed Computing, 16th International Conference (DISC 2002)*, pages 17–32, 2002.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[GMS08] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In *Advances in Cryptology – EUROCRYPT 2008*, pages 289–306, 2008.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC '87)*, pages 218–229, 1987.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[Gol04] Oded Goldreich. *The Foundations of Cryptography – Volume 2, Basic Applications.* Cambridge University Press, 2004.

[GSW04] Philippe Golle, Jessica Staddon, and Brent R. Waters. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security, Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004, Proceedings*, pages 31–45, 2004.

[HAJ$^+$14] Warren He, Devdatta Akhawe, Sumeet Jain, Elaine Shi, and Dawn Song. Shadowcrypt: Encrypted web applications for everyone. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1028–1039. ACM, 2014.

[HHS08] Iftach Haitner, Jonathan J Hoch, and Gil Segev. A linear lower bound on the communication complexity of single-server private information retrieval. In *Theory of Cryptography*, pages 445–464. Springer, 2008.

[HILM02] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*, pages 216–227, 2002.

[HKK12] Kevin W. Hamlen, Lalana Kagal, and Murat Kantarcioglu. Policy enforcement framework for cloud data management. *IEEE Data Eng. Bull.*, 35(4):39–45, 2012.

[HKS$^+$10] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Tasty: Tool for automating secure two-party computations. *IACR Cryptology ePrint Archive*, 2010:365, 2010.

[HL10] Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols – Techniques and Constructions.* Information Security and Cryptography. Springer, 2010.

[HLMR11] Martin Hirt, Christoph Lucas, Ueli Maurer, and Dominik Raub. Graceful degradation in multi-party computation (extended abstract). In *5th International Conference on Information Theoretic Security (ICITS 2011)*, pages 163–180, 2011.

[IKK12] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC '08)*, pages 433–442, New York, NY, USA, 2008. ACM.

[Ins97] Alfred Inselberg. Multidimensional detective. In *Information Visualization, 1997. Proceedings., IEEE Symposium on*, pages 100–107. IEEE, 1997.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer—efficiently. In *Advances in Cryptology – CRYPTO 2008*, pages 572–591, 2008.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88)*, pages 20–31, 1988.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography.* CRC Press, 2014.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 364–364. IEEE Computer Society, 1997.

[KO13] Kaoru Kurosawa and Yasuhiro Ohtaki. How to update documents verifiably in searchable symmetric encryption. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *CANS 13*, volume 8257 of *LNCS*, pages 309–328, Paraty, Brazil, November 20–22, 2013. Springer, Berlin, Germany.

[KP13] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In Ahmad-Reza Sadeghi, editor, *FC 2013*, volume 7859 of *LNCS*, 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*, pages 965–976, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[Kur14] Kaoru Kurosawa. Garbled searchable symmetric encryption. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 234–251, Christ Church, Barbados, March 3–7, 2014. Springer, Berlin, Germany.

[LCS$^+$14] Billy Lau, Simon Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva. Mimesis aegis: A mimicry privacy shield–a systems approach to data privacy on public cloud. In *Proceedings of the 23rd USENIX conference on Security Symposium*, pages 33–48. USENIX Association, 2014.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[LYCL11] Ming Li, Shucheng Yu, Ning Cao, and Wenjing Lou. Authorized private keyword search over encrypted data in cloud computing. In *2011 International Conference on Distributed Computing Systems, ICDCS 2011, Minneapolis, Minnesota, USA, June 20-24, 2011*, pages 383–392, 2011.

[MH81] Ralph C. Merkle and Martin E. Hellman. On the security of multiple encryption. *Commun. ACM*, 24(7):465–467, 1981.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay – secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms (SODA 2001)*, pages 448–457, 2001.

[NPG14] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*, pages 639–654, Berkeley, California, USA, May 18–21, 2014. IEEE Computer Society Press.

[OKKM13] Wakaha Ogata, Keita Koiwa, Akira Kanaoka, and Shin'ichiro Matsuo. Toward practical searchable symmetric encryption. In Kazuo Sakiyama and Masayuki Terada, editors, *IWSEC 13*, volume 8231 of *LNCS*, pages 151–167, Okinawa, Japan, 2013. Springer, Berlin, Germany.

[PGFW14a] Jason Perry, Debayan Gupta, Joan Feigenbaum, and Rebecca N. Wright. The secure computation annotated bibliography, 2014. `http://paul.rutgers.edu/~jasperry/ssc-annbib.pdf`.

[PGFW14b] Jason Perry, Debayan Gupta, Joan Feigenbaum, and Rebecca N. Wright. Systematizing secure computation for research and decision support. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 380–397. Springer International Publishing, 2014.

[PKV$^+$14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos Keromytis, and Steve Bellovin. Blind seer: A scalable private dbms. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 359–374. IEEE, 2014.

[PRZB11] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC '04)*, pages 242–251, 2004.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT 2009*, pages 250–267, 2009.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer Berlin Heidelberg, 2008.

[Rab81] Michael O. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard University, 1981.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC '89)*, pages 73–85, 1989.

[Riv99] Ronald Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. *Unpublished manuscript*, 1999.

[Seg13] Aaron Segal. Comparison of SMPC platforms. Unpublished manuscript, 2013.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[SPS14] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS 2014*, San Diego, California, USA, February 23–26, 2014. The Internet Society.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55, 2000.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1982)*, pages 160–164, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1986)*, pages 162–167, 1986.

[YZW06] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-preserving queries on encrypted data. In *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings*, pages 479–495, 2006.