The Dissertation Committee for Justin Lee Brickell
certifies that this is the approved version of the following dissertation:

# Privacy-Preserving Computation for Data Mining

Committee:

Vitaly Shmatikov, Supervisor

Inderjit Dhillon

Adam Klivans

Emmett Witchel

Rebecca Wright

# Privacy-Preserving Computation for Data Mining

by

**Justin Lee Brickell, B.A.; B.S.; M.S.**

**DISSERTATION**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**DOCTOR OF PHILOSOPHY**

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2009

## Acknowledgments

I am grateful to Vitaly Shmatikov for all of the advice that he has given me. Without his criticism of my writing, I might never have learned how to present my work clearly to the research community. Without his encouragement, I would have abandoned many projects that at times seemed hopeless, but all of which turned out to bear fruit just like he said they would. He has always found the time to answer my questions and to offer help when I need it.

I am thankful to the members of my committee, who provided me with useful feedback during my proposal, and who did not shy away from asking tough questions about my research. I am particularly grateful to Inderjit Dhillon, who served as my advisor in the years before I firmly settled on security and privacy as my research area. The data mining work that I did with him proved to be very helpful when I turned my attention to privacy problems that are of interest to the data mining community.

My father, Ernest Brickell, is an enormous influence on my academic career. His research is in cryptography and security, and I entered graduate school fully intending to blaze my own path and choose a different field. My plan fell apart when I took Vitaly's security course my second year, and I soon found myself at cryptography conferences being asked "are you by any chance related to *Ernie* Brickell?" Even though I initially resisted the idea, it has been a pleasure to follow in my father's footsteps.

It has been a pleasure to be a member of the graduate student community at UT. Among its members are both colleagues and many dear friends.

My wife, Carol Chen, has supported me unconditionally and has made sure that I pay enough attention to the rest of my life outside of research.

# Privacy-Preserving Computation for Data Mining

Publication No. __________________

Justin Lee Brickell, Ph.D.
The University of Texas at Austin, 2009

Supervisor: Vitaly Shmatikov

As data mining matures as a field and develops more powerful algorithms for discovering and exploiting patterns in data, the amount of data about individuals that is collected and stored continues to rapidly increase. This increase in data heightens concerns that data mining violates individual privacy. The goal of data mining is to derive aggregate conclusions, which should not reveal sensitive information. However, the data-mining algorithms run on databases containing information about individuals which may be sensitive. The goal of privacy-preserving data mining is to provide high-quality aggregate conclusions while protecting the privacy of the constituent individuals.

The field of "privacy-preserving data mining" encompasses a wide variety of different techniques and approaches, and considers many different threat and trust models. Some techniques use *perturbation*, where noise is added (either directly to the database that is the input to the algorithm or to the output of queries) to obscure values of sensitive attributes; some use *generalization*, where identifying attributes are given less specific values; and some use *cryptography*, where joint computations between multiple parties are performed on encrypted data to hide inputs. Because these approaches are applied to different scenarios with different threat models, their overall effectiveness and privacy properties are incomparable.

In this thesis I take a pragmatic approach to privacy-preserving data mining and attempt to determine which techniques are suitable to real-world problems that a data miner might wish to solve, such as evaluating and learning decision-tree classifiers. I show that popular techniques for sanitizing databases prior to publication either fail to provide any meaningful privacy guarantees, or else degrade the data to the point of having only negligible data-mining utility.

Cryptographic techniques for secure multi-party computation are a natural alternative to sanitized data publication, and guarantee the privacy of inputs by performing computations on encrypted data. Because of its heavy reliance on public-key cryptography, it is conventionally thought to be too slow to apply to real-world problems. I show that tailor-made protocols for specific data-mining problems can be made fast enough to run on real-world problems, and I strengthen this claim with empirical runtime analysis using prototype implementations. I also expand the use of secure computation beyond its traditional scope of applying a known algorithm to private inputs by showing how it can be used to efficiently apply a private algorithm, chosen from a specific class of algorithms, to a private input.

# Table of Contents

# Chapter 1

# Introduction

## 1.1 Privacy-Preserving Data Mining

*Data mining*, or *knowledge discovery*, is the act of processing massive amounts of data in order to learn small amounts of useful information. For example, through the analysis of product-purchase transactions, a company may determine that two products are highly correlated: consumers who purchased product “A” were also likely to purchase product “B.” This type of information can have significant commercial value. As the commercial benefits of data mining have increased and the costs of storing and processing data have decreased, there has been an explosion in the amount of data that is collected and stored about individuals. For the most part consumers have benefitted from the mining of their data, as it has allowed companies to more precisely target their advertising and to provide consumers with uniquely tailored recommendations. However, the massive-scale mining of customer data also creates security and privacy concerns. The demand for combining data from multiple sources to facilitate data mining has given rise to companies such as ChoicePoint and Acxiom, who are in the business of aggregating and reselling data about consumers. This creates a single point of failure where a breach of security at one of these companies can compromise the privacy of thousands of consumers. A security breach at ChoicePoint in 2005 resulted in the theft of information about more than 163,000 consumers [28], and data breaches at Acxiom affected as many as 1.6 billion records [106]. Security breaches of this scale could be avoided with the development of techniques that allow data mining over aggregated databases without the need to actually combine the data in a single location.

In addition to the accidental loss of customer data due to theft, there have been several recent instances of companies intentionally releasing customer data for the express purpose of exploratory data mining by third parties. For example, AOL released a dataset for the academic community, and Netflix released a dataset so that they could sponsor a prize for the best data-mining algorithm [61,62]. In both cases, the data was supposed to be anonymous and unlinkable, because obvious identifiers such as names were removed. However, it was later shown that it was far easier to link individuals with their "anonymous" records than the companies intended [93].

There are also many real-world situations where one party may need to perform a computation on a single individual's private data. For example, a company may be able to diagnose software errors based on information about a customer's computer at the time of the software fault. However, customers will be unable to take advantage of these sorts of services if they are unwilling to disclose their private data.

In collaborative medical applications, multiple hospitals may wish to run a data-mining algorithm on a joint database which combines all of their data. The larger database provides better results by increasing the size of the training set. However, the hospitals may be prevented from sharing their data directly by legal requirements or by their own desire to protect their intellectual property.

If a company wishes to run data-mining algorithms on a database in its possession, it may be sufficient to provide customer privacy by enacting a data handling policy (*e.g.*, no customer data on laptops) or by using conventional cryptographic techniques. Scenarios in which multiple parties with privacy concerns participate in a data-mining process, on the other hand, require the development of new techniques. It is the goal of research in privacy-preserving data mining to develop techniques which enable these sorts of scenarios.

## 1.2 What is Privacy?

A privacy-preserving data-mining scenario can have many different parties, including several individuals (such as customers) who contribute their data to a database, a database owner who controls access to the database, and a data miner who runs a learning algorithm on the database. Since there are a wide variety of possible scenarios with multiple parties, it is meaningless to make the unqualified statement that a data-mining protocol is "private." Instead, we must explicitly state *what* information is being kept private, *who* owns the information, and *what* the adversary is capable of. After making these determinations, we can construct rigorous definitions of privacy that are scenario-appropriate. Here we examine some types of information that might be kept private in a data-mining scenario.

### 1.2.1 Privacy of the data

Typically the data are in the possession of a database owner who wants to ensure privacy on behalf of the individual contributors. (One exception is the "Data Collection" scenario, in which the individual contributors desire privacy *from* the database owner. See section 3.1.) The adversary is the data miner, who is supposed to learn as little sensitive information about individuals as possible while at the same time learning a useful data-mining result. The data miner may learn private information about individuals in one of two ways. He may be able to draw private conclusions about individuals from the output of the data-mining algorithm, or he may perform additional analysis on the data that was provided to him in order to tease out sensitive information beyond that found in the data-mining output.

**Privacy loss from the data-mining result.** In some cases, individual privacy loss can stem from the data-mining result itself (even though data-mining results tend to be aggregate conclusions). This is especially the case when the adversary has background information about the individual. Consider, for instance, an association rule that tells us that people with high blood pressure have a high risk of heart disease. If an individual considers her heart disease to be private information, and a data miner knows that this individual has high blood pressure, then the association rule combined with the data miner's background knowledge gives the data miner information about her private information.

Ideally, we could say that a data-mining result is privacy-preserving if adversaries can learn no more about individuals when they have access to the result than they can without access to the result. Unfortunately, it is impossible to achieve such a definition because it would require accounting for all possible auxiliary information that the adversary may have access to [38].

In fact, if we allow for arbitrary forms of auxiliary information, a data-mining result could leak information about an individual who isn't even in the database. Consider, for instance, an adversary who knows in advance that a certain individual is of average height. If the average height is published, then the adversary learns the height of the individual. The concept of $\epsilon$-*differential privacy* [38], discussed in Section 3.4, says that a data-mining result is private if it changes only very slightly when a single individual is removed or added to the database. While it is possible that a data-mining result could still leak a substantial amount of information about an individual, it would leak essentially the same amount of information about her even if she were not in the database.

**Privacy loss from intermediate data.** Even if the result of a data-mining algorithm doesn't violate the privacy of individual contributors, it is possible that during the course of computing the result the data miner will learn additional information which does violate their privacy. For example, if data mining requires the database owner to transmit the entire database to the data miner, then the data miner will learn much more about individuals' private records than he would learn from the data-mining result alone. This may still be the case if the database is sanitized before transmission, if the sanitization is inadequate or fails to account for all of the data miner's background knowledge.

If data mining takes place as an interactive process (as in the secure multi-party computation framework), then intermediate data consists of the messages that are passed back and forth between participants. An adversarial data miner may attempt to analyze these messages in order to draw conclusions other than those conclusions which are a consequence of the data-mining output.

### 1.2.2 Privacy of the algorithm and its output

The data miner may wish to keep the data-mining algorithm that he applies to the database private from the database owner. This could either be because the algorithm is proprietary, or because the data miner does not wish to reveal anything about the type of conclusion he is hoping to draw from the data. From a theoretical standpoint, there is no difference between an interactive scenario where one party provides a private function to be applied to the other party's private data, and a scenario in which both parties provide private inputs to a public function. They are theoretically equivalent because the public function could be a universal circuit, which accepts a description of a function as one of its inputs. However, from a practical standpoint it is very useful to provide protocols for private-algorithm scenarios that are more efficient than the privacy-preserving simulation of a universal circuit.

## 1.3 This Thesis

The goal of this thesis is to answer the following question:

**How can we continue to support popular data mining tasks while minimizing the risks of collecting, warehousing, and distributing the private data that is needed as input?**

We develop methods that (1) are **efficient**, so that they can be utilized in real-world systems; (2) provide **meaningful data-mining utility**, so that the privacy-protection techniques do not reduce the quality of data-mining results; and (3) satisfy **meaningful definitions of privacy** with **provable guarantees**, so that the threat model and allowed leakage of sensitive information are precisely understood.

The remainder of this thesis is organized as follows. Chapter 2 provides background about notation and cryptographic techniques that are used throughout the thesis. Chapter 3 gives related work concerning the many different approaches to privacy-preserving data-mining. Chapters 4–8 contain the technical contribution of this thesis. At the highest level, they are divided between Chapter 4 which is about *collecting* private data, and Chapters 5–8 which are about *using* private data.

Within Chapters 5–8, there is an additional division between Chapter 5, which shows how popular sanitized data publishing techniques are *ineffective*, and Chapters 6–8 which give several *effective* secure multi-party computation protocols for data-mining problems. Specifically, Chapter 6 examines graph algorithms (such as all pairs shortest path), Chapter 7 examines the evaluation of branching programs, and Chapter 8 examines decision-tree learning. Finally, Chapter 9 concludes the thesis.

This thesis contains previously published work [17–21].

# Chapter 2

# Preliminaries and Tools

In this chapter, we define cryptographic concepts and notation that are used throughout the thesis.

## 2.1 Public-Key Encryption

We take the standard definition of a public-key cryptosystem from [12]. A public-key (or "asymmetric") encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, as follows:

- A randomized *key generation* algorithm $\mathcal{K}$ which returns a pair $(x, y)$ of keys. These are the private and public keys, respectively.
- A (possibly randomized) *encryption* algorithm $\mathcal{E}$, which takes a public key $y$ and a plaintext $m$, and returns a ciphertext $C$. To denote encryption we will write

$$C = \{m\}_y.$$

- A deterministic *decryption* algorithm $\mathcal{D}$, which takes a private key $x$ and a ciphertext $C$ to return a message $m$. We will denote this as

$$m = \mathrm{dec}_x(C).$$

We require that $\text{dec}_x(\{m\}_y) = m$. This means that the owner of the private key $x$ can recover a message encrypted with the public key $y$.

Many times in this thesis we will need to serially encrypt a text under multiple public keys $y_N, ..., y_i$, and we will use the following notation to make this less cumbersome:

$$\{m\}_{y_N:y_i} \overset{\text{def}}{=} \{...\{\{m\}_{y_N}\}_{y_{N-1}}...\}_{y_i}.$$

### 2.1.1 Indistinguishability under adaptive chosen ciphertext attack

We require that the cryptosystems used in our protocols have the property of *indistinguishability under adaptive chosen ciphertext attack (IND-CCA2)* [12]. Intuitively, this means that it is impossible to learn any information a plaintext $m$ from an encryption $\{m\}_y$, even when given access to a decryption oracle for all ciphertexts other than $\{m\}_y$. Although IND-CCA2 is a very strong property, there are cryptosystems that are known to satisfy it [15, 30]. Under the RSA assumption and in the random-oracle model, RSA with OAEP padding satisfies IND-CCA2 [49].

In our definition of an IND-CCA2 encryption scheme, we will make use of the *distinguishing game*.

#### The distinguishing game

The distinguishing game is played between a challenger and an oracle. This game is slightly different from, but equivalent to, the standard *adaptive chosen ciphertext game* [12].

1. The oracle chooses a keypair $(x, y)$, and gives $y$ to the challengers.

2. The challenger may encrypt polynomially many messages $m$ using $y$. The challenger may also choose polynomially many ciphertexts $C$ and send them to the oracle, who sends back $\mathrm{dec}_x(C)$.

3. The challenger chooses two plaintexts $m_0$ and $m_1$.

4. The oracle chooses a bit $b \in \{0, 1\}$ uniformly at random, and returns the ordered pair $(\{m_b\}_y, \{m_{\bar{b}}\}_y)$.

5. The challenger may encrypt polynomially many messages $m$ using $y$. The challenger may also choose polynomially many ciphertexts $C \neq \{m_b\}_y, \{m_{\bar{b}}\}_y$ and send them to the oracle, who sends back $\mathrm{dec}_x(C)$.

6. The challenger attempts to guess whether $b = 0$ or $b = 1$.

Let $A$ be a polynomial-time challenger. Then

$$\Pr[A(m_0, m_1, 0) = 1]$$

is the probability that $A$ outputs 1 when the bit $b = 0$, and

$$\Pr[A(m_0, m_1, 1) = 1]$$

is the probability that $A$ outputs 1 when the bit $b = 1$. In both cases, the probability is taken over the randomness of the key generation in step 1. The challenger's *advantage* is equal to

$$\Pr[A(m_0, m_1, 1) = 1] - \Pr[A(m_0, m_1, 0) = 1].$$

Now we can define indistinguishability under adaptive chosen ciphertext attack as follows:

**Definition 1.** *A cryptosystem is IND-CCA2 if, for all probabilistic polynomial-time challengers, the advantage in the distinguishing game is negligible (dominated by $\frac{1}{f(\rho)}$, where $f$ is any polynomial and $\rho$ is a security parameter).*

## 2.2 Digital Signatures

We use the standard definition of digital signature schemes from [12]. A *digital signature scheme* $\mathcal{DS} = (\mathcal{K}, \textsc{sig}, \mathrm{VF})$ consists of three algorithms:

- The randomized *key generation* algorithm $\mathcal{K}$, which returns a pair $(u, v)$ of keys. These are the private and public keys, respectively.
- The (possibly randomized) *signing algorithm* $\textsc{sig}$, which takes a private key $u$ and a message $m$ to produce a signature $\sigma = \textsc{sig}_u\{m\}$.
- The deterministic *verification* algorithm VF, which takes a public key $v$, a message $m$, and a candidate signature $\sigma$. VF returns 1 if $\sigma$ is a valid signature of $m$ with key $u$, and 0 otherwise. That is, $\mathrm{VF}(v, m, \textsc{sig}_u\{m\})$ returns 1.

The desired security property for a digital signature scheme is *unforgeability*, which means that without the private key $u$, it is computationally infeasible to produce a signature $\textsc{sig}_u(m)$ for a message $m$ that one has not previously seen signed with $u$. For a more formal treatment, see [12].

## 2.3 Oblivious Transfer

*Oblivious transfer* (OT) is a fundamental cryptographic primitive [70, 104]. A 1-out-of-$N$ oblivious transfer, denoted as $OT_N^1$, is a protocol between two parties, the Chooser and the Sender. The Chooser's input is the index $\sigma \in \{0, ..., N-1\}$. The Sender's inputs are the values $M_0, M_1, ..., M_{N-1}$. As a result of the protocol, the Chooser learns $M_\sigma$ (and *only* $M_\sigma$), while the Sender learns nothing.

In our constructions, we use oblivious transfer as a "black-box" primitive, *i.e.*, our constructions do not depend on a particular OT implementation. In our implementations, we employ the Naor-Pinkas constructions for $OT_2^1$ [90]. Below, we restate Protocol 3.1 from [90]. It performs some precomputation so that the amortized cost of each 1-out-of-$N$ oblivious transfer is a single modular exponentiation. In the following, $H$ is a hash function that is modeled as a random oracle, and $g$ is a generator for the group $\mathbb{Z}_q$ of prime order. All mathematics is done modulo $\mathbb{Z}_q$.

**Initialization:** The sender chooses $N-1$ random constants $C_1, C_2, ..., C_{N-1}$. It also chooses a random $r$ and computes $g^r$. The values $C_1, ..., C_{N-1}$ and $g^r$ are sent to the chooser and play the role of the public key of the sender. The same values will be used for all transfers. The sender precomputes for every $1 \leq i \leq N-1$ the value $(C_i)^r$.

**Transfer:** The sender's input is $M_0, M_1, ..., M_{N-1}$. The chooser's input is $\sigma \in \{0, ..., N-1\}$ (she should learn $M_\sigma$).

- The chooser selects a random $k$ and sets $PK_\sigma = g^k$. If $\sigma \neq 0$ she computes $PK_0 = C_\sigma / PK_\sigma$. She sends $PK_\sigma$ to the sender and can already compute a decryption key $(g^r)^k = (PK_\sigma)^r$.

- The sender computes $(PK_0)^r$ and then for every $1 \leq i \leq N-1$ computes (without doing any additional exponentiations)

$$(PK_i)^r = (C_i)^r / (PK_0)^r.$$

  The sender chooses a random string $R$. He then encrypts each $M_i$ by computing $H((PK_i)^r, R, i) \oplus M_i$, and sends these encryptions and $R$ to the chooser.

- The chooser uses $H((PK_\sigma)^r, R, \sigma)$ to decrypt $M_\sigma$.

## 2.4 Homomorphic Encryption

A *homomorphic encryption scheme* is a semantically secure cryptosystem that permits algebraic manipulations on plaintexts given their respective ciphertexts. In this thesis, we require an encryption scheme with an *additively homomorphic* property, which allows $\{c_1 + c_2\}_y$ to be computed from $\{c_1\}_y$

and $\{c_2\}_y$ without knowledge of the private key $x$ corresponding to the public key $y$.

In our prototype implementation, we use the Paillier cryptosystem [101], which operates as follows:

**Key Generation:** Large primes $p$ and $q$ are chosen independently at random, and $\mathcal{N} = pq$ and $\lambda = \mathsf{lcm}(p-1, q-1)$ are computed. Let $\mathcal{G} \in \mathbb{Z}^*_{\mathcal{N}^2}$ be chosen randomly, and then compute

$$\mu = \left(\frac{((\mathcal{G}^\lambda \mod \mathcal{N}^2) - 1)}{\mathcal{N}}\right)^{-1}.$$

If the multiplicative inverse does not exist, a new $\mathcal{G}$ must be chosen. The public key is $PK = (\mathcal{N}, \mathcal{G})$ and the secret key is $SK = (\lambda, \mu)$.

**Encryption:** To encrypt a message $m$, first a random value $r \in \mathbb{Z}^*_{\mathcal{N}^2}$ is chosen. Then compute the ciphertext as $c = \mathcal{G}^m \cdot r^{\mathcal{N}} \mod \mathcal{N}^2$.

**Decryption:** For a ciphertext $c$, compute

$$m = \left(\frac{(c^\lambda \mod \mathcal{N}^2) - 1}{\mathcal{N}}\right) \cdot \mu \mod \mathcal{N}$$

**Homomorphic encryption:** Paillier is additively homomorphic, so that the product of two ciphertexts will decrypt to the sum of their plaintexts:

$$D[E[m_1, r_1] \cdot E[m_2, r_2] \mod \mathcal{N}^2] = m_1 + m_2 \mod \mathcal{N}.$$

The Paillier cryptosystem is semantically secure under the Decisional Composite Residuosity assumption [101].

## 2.5 Secure Multi-Party Computation

### 2.5.1 Garbled circuits

*Garbled circuits* are a fundamental technique in secure multi-party computation. Originally proposed by Yao [132], the garbled circuits method enables secure constant-round computation of any two-party functionality. We only give a brief overview here; a detailed explanation can be found in [80].

Let $C$ be a boolean circuit which receives two $n$-bit inputs $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_n$, and outputs bit $C(x, y) \in \{0, 1\}$. (If the circuit takes only one input, we denote the other input as $\perp$.) Consider Alice and Bob who wish to securely compute $C(x, y)$, where $x$ is Alice's input, $y$ is Bob's input. Yao's method transforms any $C$ into a secure *garbled* circuit $C'$, which enables computation of $C(x, y)$ without revealing $x$ to Bob or $y$ to Alice.

For each wire $i$ of the circuit, Alice generates two random *wire keys* $w_i^0$ and $w_i^1$. These wire keys are used as labels encoding, respectively, 0 and 1 on that wire. Now consider a single gate $g$ in $C$, described by some boolean function $g : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$. Let the two input wires to $g$ be labeled $A$ and $B$, and let the output wire be labeled $C$. The corresponding wire keys are $w_A^0, w_A^1, w_B^0, w_B^1, w_C^0, w_C^1$.

The garbled gate $g'$ of circuit $C'$ is defined by a random permutation of the following four ciphertexts, where $\{x\}_\kappa$ is a symmetric-key encryption of plaintext $x$ under key $\kappa$ (see [80] for the properties that the encryption scheme must satisfy).

$$\begin{array}{rclrcl} c_{00} & = & \{\{w_C^{g(0,0)}\}_{w_B^0}\}_{w_A^0} & c_{01} & = & \{\{w_C^{g(0,1)}\}_{w_B^0}\}_{w_A^1} \\ c_{10} & = & \{\{w_C^{g(1,0)}\}_{w_B^1}\}_{w_A^0} & c_{11} & = & \{\{w_C^{g(1,1)}\}_{w_B^1}\}_{w_A^1} \end{array}$$

Alice garbles all gates of the circuit in this manner, and sends the entire garbled circuit to Bob.

Garbled circuit evaluation proceeds as follows. For each input wire $i$ associated with Alice, Alice simply sends to Bob the wire key $w_i^{b_A}$ encoding Alice's input bit $b_A$ on that wire. This leaks no information about the value of $b_A$ because the wire keys are random. For each input wire $j$ associated with Bob, Alice and Bob engage in $OT_2^1$ protocol. Alice's inputs as the sender are the two wire keys $w_j^0$ and $w_j^1$, and Bob's input as the chooser is his bit $b_B$ on that wire. As a result of the OT protocol, Bob learns the wire key $w_j^{b_B}$ encoding his input without revealing $b_B$ to Alice.

Bob evaluates the circuit starting from the gates where he has a wire key for each input wire. For each such gate, Bob can decrypt exactly one of the four ciphertexts, and learn the key $w_C$ encoding the value of the gate's output wire. If the output wire is used as an input into another gate, Bob continues the process. This evaluation procedure maintains the invariant that, for each circuit wire $i$, Bob learns exactly one wire key $w_i^b$. This wire key is random and thus leaks no information about the bit $b$ it "represents."

In the standard Yao's method, Alice provides a mapping for the wire keys $w_{\text{out}}^0$ and $w_{\text{out}}^1$ of each circuit output wire *out* to 0 and 1, respectively.

This allows circuits transformed by Yao's method to be used as "black boxes" which have the same functionality as normal circuits, but hide the parties' respective inputs.

In some of our constructions, we use Yao's garbled circuits in a non-black-box way to implement a *conditional oblivious transfer*. In our modification, Alice does not provide a mapping from $w^0_{\text{out}}$ and $w^1_{\text{out}}$ to 0 and 1; instead, we consider $w^0_{\text{out}}$ or $w^1_{\text{out}}$ to be Bob's final output from evaluating the circuit. Furthermore, $w^0_{\text{out}}$ and $w^1_{\text{out}}$ can be arbitrary strings of our choosing rather than random strings. Using Yao's method in this non-standard way, Bob learns exactly one of two values depending on the output of the function encoded by the circuit.

We use a similar modification to implement a conditional oblivious transfer for 1-out-of-$m$ values (as opposed to 1-out-of-2). We encode the condition logic using a Yao circuit which has $\log_2 m$ output wires to encode the $m$ different possible output values. As in the standard Yao's construction, each wire has two random keys associated with it, representing, respectively, 0 and 1. These random keys are used to encrypt a table with $m$ randomly permuted rows (observe that there is a 1:1 correspondence between the rows, all possible values of $a$, and all possible combinations of bit values on the $\log_2 m$ output wires). For each value of $a$, the encrypted row contains the index of and the decryption key for the appropriate next node in the evaluation, encrypted under the output-wire keys corresponding to the bit representation of $a$.

For instance, suppose that $m = 4$, so that each attribute takes values from 0 to 3, and thus each internal node in the tree has 4 children. We represent each node by a gate with two output wires, $w_0, w_1$. Let $w_i^0$ and $w_i^1$ be the random keys representing, respectively, 0 and 1 values for wire $i$. If the bit representation of $a$ is $\alpha\beta$, then evaluating this gate reveals to the evaluator $w_0^\alpha$ and $w_1^\beta$. Note that the evaluator does not learn $a$.

Let $h_a$ be the string containing the index and the decryption key for the child node corresponding to the attribute value $a$. The gate is accompanied by a random permutation of the following 4 ciphertexts: $\{\{h_0\}_{w_0^0}\}_{w_0^1}$, $\{\{h_1\}_{w_0^1}\}_{w_1^0}$, $\{\{h_2\}_{w_1^0}\}_{w_1^0}$, $\{\{h_3\}_{w_1^1}\}_{w_1^1}$. Observe that the keys $w_0^\alpha$ and $w_1^\beta$ decrypt exactly one row of this table, namely, the row corresponding to $a$. By decrypting it, the evaluator learns exactly one of the $m$ values.

Our prototype implementations make use of two implementations of Yao's method: the Fairplay implementation [82] and the Wisconsin implementation [72]. We refer to it as the YAO subroutine, which, on input of a two-party functionality, produces its garbled-circuit implementation.

### 2.5.2 Definitions of privacy

**Privacy against semi-honest adversaries.** We use a simplified form of the standard definition of security in the static semi-honest model due to Goldreich [55] (this is the same definition as used, for example, by Lindell and Pinkas [78]).

**Definition 2.** *(computational indistinguishability): Let $S \subseteq \{0,1\}^*$. Two ensembles (indexed by $S$), $X \overset{\text{def}}{=} \{X_w\}_{w \in S}$ and $Y \overset{\text{def}}{=} \{Y_w\}_{w \in S}$ are* computationally indistinguishable (by circuits) *if for every family of polynomial-size circuits, $\{D_n\}_{n \in \mathbb{N}}$, there exists a negligible (*i.e.*, dominated by the inverse of any polynomial) function $\mu : \mathbb{N} \mapsto [0,1]$ so that*

$$|\Pr[D_n(w, X_w) = 1] - \Pr[D_n(w, Y_w) = 1]| < \mu(|w|)$$

*In such a case we write $X \overset{c}{\equiv} Y$.*

Suppose $f$ is a polynomial-time functionality (deterministic in all cases considered in this thesis), and $\pi$ is the protocol. Let $x$ and $y$ be the parties' respective private inputs to the protocol. For each party, define its *view* of the protocol as $(x, r^1, m_1^1, \ldots, m_k^1)$ (respectively, $(y, r^2, m_1^2, \ldots, m_l^2)$), where $r^{1,2}$ are the parties' internal coin tosses, and $m_j^i$ is the $j^{th}$ message received by party $i$ during the execution of the protocol. We will denote the $i^{th}$ party's view as $\mathsf{view}_i^\pi(x,y)$, and its output in the protocol as $\mathsf{output}_i^\pi(x,y)$.

**Definition 3.** *Protocol $\pi$ securely computes deterministic functionality $f$ in the presence of static semi-honest adversaries if there exist probabilistic polynomial-time simulators $S_1$ and $S_2$ such that*

$$\begin{array}{rcl} \{S_1(x, f(x,y))\}_{x,y \in \{0,1\}^*} & \overset{c}{\equiv} & \{\mathsf{view}_1^\pi(x,y)\}_{x,y \in \{0,1\}^*} \\ \{S_2(y, f(x,y))\}_{x,y \in \{0,1\}^*} & \overset{c}{\equiv} & \{\mathsf{view}_2^\pi(x,y)\}_{x,y \in \{0,1\}^*} \end{array}$$

*where $|x| = |y|$.*

Informally, this definition says that each party's view of the protocol can be efficiently simulated given only its private input and the output of the algorithm that is being computed (and, therefore, the protocol leaks no information to a semi-honest adversary beyond that revealed by the output of the algorithm).

**Privacy against malicious and covert adversaries.** While semi-honest adversaries are required to faithfully execute protocols as specified, *malicious* adversaries [54] may arbitrarily deviate from protocol specification, and may even formulate their deviations online in response to earlier messages. In general, protocols that are secure in the semi-honest model can be transformed into protocols that are secure in the malicious model at a constant cost under certain number-theoretic assumptions [66, 79, 125]. The transformation involves requiring that the parties commit to their inputs, and then prove (in zero-knowledge) that all messages are consistent with the committed input and the messages that have already been received in the protocol transcript. For real-world scenarios, this generic transformation between semi-honest and malicious protocols is very costly, so there is interest in finding malicious model protocols that are more efficient. For instance, the protocol that we provide in Chapter 4 is secure in the malicious model, but does not require the use of zero-knowledge proofs.

Sitting between the semi-honest and malicious adversaries in strength is the *covert* adversary [8, 58]. Like a malicious adversary, a covert adversary can deviate arbitrarily from the protocol specification. However, a covert adversary wishes to avoid being caught doing so. A protocol that is secure against a covert adversary guarantees that if the privacy of honest participants is broken, then with high probability the dishonest behavior that caused the privacy breach is detected. Yao's garbled circuit protocol can be made secure against covert adversaries using the "cut-and-choose" techniques [58], where he circuit sender produces $n$ different versions of the garbled circuit, and then reveals $n-1$ of them (chosen by the evaluator) to prove that they are correctly formed.

# Chapter 3

# Background and Related Work

Research in privacy-preserving data mining has a long history that precedes the popularization of the term “data mining.” Early work on database privacy examined the problem of databases that answered COUNT and SUM queries provided by users, where COUNT queries returned the number of records which satisfied a predicate, and SUM queries summed over a field for those records which satisfied a predicate. Although these queries in isolation do not reveal individual record values, researchers observed that an adversary could analyze the intersection of several queries in order to tease out sensitive information. Research in the late 70s and early 80s attempted to solve this problem by either adding noise to the queries, or by restricting the types of queries that are allowed [1,33,36]. The definitions of privacy in this early work seem weak by modern standards, since they are only concerned with attacks that reveal a sensitive value with absolute certainty.

Privacy-preserving data-mining research from the late 90s until today has greatly expanded the scope of the database privacy research, by considering different database access scenarios, different types of adversarial abilities, and different privacy goals. There are three main methodologies commonly used in current research, each of which has different and largely incomparable objectives. The work of Sweeney [112] is concerned with the re-identification of supposedly anonymous database records by adversaries who link records with identities using *quasi-identifiers*—attributes such as age, sex, and zip code which may be easily found in external databases. Work of this flavor uses the *generalization and suppression* of quasi-identifiers to make these linkage attacks more difficult. The work of Agrawal and Srikant [6] is concerned that adversaries may learn the exact values of sensitive attributes, so they add *random noise* to the data in order to mask true values while still preserving aggregate statistics that are useful to data mining. The work of Lindell and Pinkas [78] examines privacy-preserving data mining from a multi-party computation perspective, in which two parties want to run a data-mining algorithm on a joint database without revealing their own portions of the database to each other. They apply cryptographic techniques from the area of *secure multi-party computation* to execute data-mining algorithms in a manner that prevents any information other than the algorithm output from being revealed to the parties.

The field of privacy-preserving data mining is a vast landscape encompassing a wide variety of problems, each of which has different trust models, goals, and scenarios. In this section, we examine work on several popular privacy-preserving data-mining problems and techniques.

## 3.1 Data Collection

The data collection problem is concerned with how a data miner obtains data in the first place. Unlike almost all other work in privacy-preserving data mining, it does not presuppose the existence of a database. The assumption is that the data miner is untrusted, and that individuals providing their data to the data miner are concerned about the data miner learning their private information. There are two basic approaches to this problem: the *unlinkability* approach, in which the data miner is provided with the exact responses of each individual, but does not know the associations between individuals and responses; and the *perturbation* approach, in which the data miner knows which individual provided which response, but the responses have been altered in some way to hide sensitive information.

The problem of collecting data so that the miner is unable to link any honest respondent to her response was investigated by Yang *et al.* [129]. Their solution is to choose $t$ of $N$ respondents as "leaders," and have respondents encrypt their responses with the leaders' public keys. Each of the leaders shuffles and rerandomizes the ciphertexts, proving to every respondent in zero-knowledge that the shuffle has been carried out correctly. This protocol requires $O(t^2)$ zero-knowledge proofs, each of which involves several rounds of communication. The protocol of Yang *et al.* does not achieve collusion resistance because if the last leader is corrupt and colludes with the data miner, they can break the anonymity of all honest respondents. This attack is due to an incorrect use of zero-knowledge proofs, and is explained in detail in Section 4.6.

So-called *mix networks* such as onion routing [34, 113] have been proposed to enable anonymous communications on public networks. Mix networks aim to hide the identities of message senders, and thus seem to be a poor match for data-mining scenarios, where the data miner may need to know exactly who the respondents are. Using a mix network to collect responses while ensuring that respondents are members of a well-defined set and that each respondent contributes no more than one response requires complicated cryptographic techniques such as group signatures, and is unlikely to be efficient for practical use.

Instead of submitting their responses anonymously within a peer group, respondents submitting sensitive data might randomly perturb their responses. In this case, privacy would be based on the inexactness of the responses, rather than on the lack of association between respondent and response. Respondents submitting data using *randomized response* [121] lie with a certain probability, so that the data miner is never certain of the truthfulness of a sensitive response. Data-mining algorithms must be modified to accept randomized response data. For instance, Du and Zhan [37] modify the popular ID3 [103] decision-tree classification algorithm.

## 3.2 Secure Multi-party Computation

In the secure multi-party computation problem, there are two data mining parties, $A$ and $B$, in possession of databases $a$ and $b$, respectively. For some data-mining functionality $f$, they wish to compute $f$ on their *joint database*; that is, they wish to compute $f(a \cup b)$. Here the union operator $\cup$ is domain-specific. Consider the case of a microdata database, which is a multi-dimensional database in which each record contains information about a single individual. The two databases $a$ and $b$ may represent a vertical partitioning (both databases contain the same records, but have different attributes for each record), horizontal partitioning (both databases have the same attributes, but contain different records), or an arbitrary partitioning. If $a$ and $b$ are weighted graphs, then the union could be a graph where each edge is the minimum weight found in $a$ or $b$.

In a slightly different secure multi-party computation scenario, party $A$ is in possession of algorithm $g$, and party $B$ holds database $y$. Together, they want to compute $g(y)$. One of the contributions of this thesis is to show that this type of "private algorithm" multi-party computation can be performed efficiently for certain data-mining tasks by trading off increased communication for decreased computation.

Within the context of data mining, the secure multi-party computation problem was first investigated by Lindell and Pinkas [78]. It is the most natural extension of the *secure multi-party computation* (SMC) paradigm to the problems encountered in data mining. In the SMC paradigm, party $P_1$

with input $x_1$ and party $P_2$ with input $x_2$ wish to compute $f(x_1, x_2)$ without revealing any more information about their inputs than is necessarily revealed by the output.

The secure multi-party computation problem is concerned only with preventing privacy loss from intermediate data. It is assumed that both parties are willing to leak whatever information about their inputs is revealed by the computation result. Since both parties run the same algorithm and learn the same result, it is also assumed that privacy of the algorithm and its output is not a concern.

Informally, security of a protocol in the SMC paradigm is defined as computational indistinguishability from some *ideal functionality*, in which a trusted third party accepts the parties' inputs and carries out the computation. The ideal functionality is thus secure by definition. The actual protocol is secure if the adversary's view in any protocol execution can be simulated by an efficient simulator who has access only to the ideal functionality, *i.e.*, the actual protocol does not leak any information beyond what is given out by the ideal functionality. Formal definitions for various settings can be found, for example, in [11, 23, 55].

Any polynomial-time multi-party computation can be done in a privacy-preserving manner using generic techniques of Yao [132] and Goldreich, Micali, and Wigderson [56]. Some SMC research has used branching programs instead of circuits as function representation [53, 89]. Generic constructions, however, are sometimes impractical due to their complexity.

Recent research has focused on finding more efficient privacy-preserving algorithms for specific problems such as computation of approximations [46], auctions [91], set matching and intersection [48], surveys [47], computation of the k-th ranked element [3] and especially data-mining problems such as privacy-preserving computation of decision trees [78], mining of association rules [67, 115], model selection [131], data imputation [64], classification of customer data [130], and mining of vertically partitioned data [43, 126].

In a typical multi-party computation scenario, when computing $f(x, y)$ securely, $f$ is assumed to be known to both parties, while $x$ and $y$ are their private inputs. In some scenarios that we consider in Chapters 7–8, we are computing $g(y)$, where $g$ is the private input of the first party and $y$ is the private input of the second party. This can be implemented by making $f$ a generic function evaluator and $x$ a description of the particular function $g$. As we show in Section 7.2.5, this approach does not scale to the size of branching programs that arise in real-world applications. Selective private function evaluation [24] considers evaluation of functions on large datasets, but the functions are much simpler than the branching programs considered in Chapter 7. To achieve practical efficiency, our protocol fundamentally relies on the structure of branching programs.

Cryptocomputing [108] has a slightly different model, evaluating $C(x)$ for one party's private circuit $C$ and another party's private circuit $x$. The result in [108] allows any bounded fan-in circuit of logarithmic depth to be evaluated privately.

In this thesis, we develop several secure multiparty computation protocols for specific functionalities which are much faster than applying the generic Yao transformation to the entire functionality. In general, we achieve this improvement in speed by trading off computation for interaction. That is, while the Yao protocol requires only two rounds of communication, our protocols have number of rounds proportional to the size of the problem. Between rounds, the parties maintain a state which is iteratively improved as the protocols progress. In some cases, this state is a consequence of the function output, and we may therefore reveal it to the parties in plaintext. For instance, in the private all-pairs shortest distance algorithm (Chapter 6), we reveal in each round the length of the next-smallest edge in the final graph. In other cases, the internal state is maintained by sharing its value between the parties, so that neither party independently has access to the state.

## 3.3 Sanitized Data Publishing

In the sanitized data publishing problem, a trusted database owner has possession of a database $D$, and he wishes to publish a *sanitized* version $D'$ which is simultaneously useful for data mining and does not compromise the privacy of the constituent individuals.

### 3.3.1 Generalization and suppression

In order to reveal less identifying information about individuals, attribute values in a database may be replaced with *generalized* values. Because the set of individuals in the population which have a particular attribute value is larger when the value is more general, this makes it more difficult for an adversary to deduce the identity of an individual in the database. For instance, a Zip code of 47605 might be replaced with 476**, or an age of 30 might be replaced by the range 30-39. In some cases attribute values are omitted entirely (replaced by "*") in which case they are said to be *suppressed*. Generalization and suppression proceed until the database satisfies some syntactic sanitization property; we describe several such properties in this section.

The $k$-anonymity property [112] and the related properties of $\ell$-diversity [81] and $t$-closeness [77] are popular syntactic database properties. The intention is that if a database satisfies these properties, then it is safe to release the database for mining. However, it appears that these definitions—although popular—are flawed, because they do not relate the satisfaction of a syntactic property with the ability (or inability) of an adversary to learn sensitive information from the published database. Our research presented in Chapter 5 supports the conclusion that these definitions are of little use. We survey several popular syntactic sanitization properties.

The $k$-anonymity property and related properties rely on the notion of a *quasi-identifier* [31]. It was famously observed [112] that removing the names (identifiers) from a medical records database was insufficient to prevent re-identification of records in the database, because the records still contained attributes such as date of birth, sex, and zip code which are also found in other databases. By linking the two databases together, it is possible to associate an identity with a record in the medical database, and then to learn sensitive information about the identified individual (such as his medical condition).

Determining which record in a database corresponds with a given individual is known as *identity disclosure*. It was later noted that it is possible to learn a sensitive attribute (such as medical condition) even when identity disclosure does not occur: this is known as *sensitive attribute disclosure*. The orthogonal attack of *membership disclosure* occurs when an adversary is able to determine whether or not an individual is included in the database (even if he cannot determine *which* record belongs to the individual).

**Notation.** Let $T = \{t_1, \ldots t_n\}$ be a data table. Each $t_i$ is a tuple of attribute values representing some individual's record. Let $\mathcal{A} = \{a_1, \ldots a_m\}$ be the set of attributes; $t[a_i]$ denotes the value of attribute $a_i$ for tuple $t$. We use the following notation for subsets of attributes and tuples. If $\mathcal{C} = \{c_1, c_2, \ldots c_p\} \subseteq A$, then $t[C]$ denotes $(t[c_1], \ldots t[c_p])$. If $U = \{u_1, u_2, \ldots u_p\} \subseteq T$, then $U[a]$ denotes $(u_1[a], \ldots u_p[a])$.

Let $\mathcal{S} \in \mathcal{A}$ be the *sensitive attribute*. This is an attribute whose value the adversary should not be able to associate with an individual (*e.g.*, medical information). Let $S = \{s_1, \ldots s_p\}$ be the set of possible attribute values for the sensitive attribute $\mathcal{S}$.

Let $\mathcal{Q} \subset \mathcal{A} \setminus \mathcal{S}$ be the *quasi-identifier*, *i.e.*, the set of non-sensitive attributes whose values may be known to the adversary for a given individual (*e.g.*, demographic information). Let $Q = \{q_1, \ldots q_p\}$ be the set of possible values for $\mathcal{Q}$. Each $q_i$ is actually a tuple of attribute values, since there are multiple attributes in the set $\mathcal{Q}$.

Two tuples $t_i$ and $t_j$ are $\mathcal{Q}$-equivalent (denoted $t_i \overset{\mathcal{Q}}{\equiv} t_j$) if $t_i[\mathcal{Q}] = t_j[\mathcal{Q}]$. This equivalence relation partitions $T$ into quasi-identifier equivalence classes, denoted as $\langle t_j \rangle$, where $t_i \in \langle t_j[\mathcal{Q}] \rangle$ iff $t_i \overset{\mathcal{Q}}{\equiv} t_j$. Let $\mathcal{E}_{\mathcal{Q}} \subseteq T$ be a set of representative records for each equivalence class imposed by $\overset{\mathcal{Q}}{\equiv}$.

Consider a subset of tuples $U = \{u_1, u_2, \ldots u_p\} \subseteq T$, and the distribution of sensitive attribute values within $U$. For any sensitive attribute value $s$, denote by $U_s$ the set $\{u \in U \mid u[\mathcal{S}] = s\}$ of tuples in $U$ whose sensitive attribute value is equal to $s$, and denote by $p(U, s)$ the corresponding fraction of tuples in $U$, computed as $\frac{|U_s|}{|U|}$. The notation $p(U, s)$ can be understood as "the probability that a randomly chosen member of $U$ has sensitive attribute value $s$."

**Syntactic Privacy Definitions.** We will briefly summarize the syntactic privacy definitions of $k$-anonymity, $\ell$-diversity, and $t$-closeness.

**$k$-anonymity.** $k$-anonymity is based on the observation that *identity disclosure* can lead to *sensitive attribute disclosure*: if an adversary can determine which database record corresponds to the target individual, then he can determine this individual's sensitive attribute value.

$k$-anonymity is a popular definition of privacy [26, 107, 112], and many approaches have been proposed for achieving it [10, 50, 63, 74, 75, 97, 102, 111, 128]. Most use a combination of domain-specific generalization and suppression on quasi-identifiers.

**Definition 4** ($k$-anonymity [107, 112])**.** *Table $T$ is $k$-anonymous if and only if for each $t_j \in \mathcal{E}_{\mathcal{Q}}$, $|\langle t_j \rangle| \geq k$*

Unfortunately, while identity disclosure will always result in sensitive attribute disclosure, preventing identity disclosure is *not* sufficient to prevent sensitive attribute disclosure [81, 114]. Consider, for instance, the original data presented in Table 3.1 and the 3-anonymous transformation of this data in Table 3.2. If the adversary knows that an individual lives in ZIP code 47602 and his age is 22, then the adversary knows that he has heart disease, even without knowing whether his record is 1, 2, or 3. Identity disclosure is prevented, but the value of the sensitive attribute is completely disclosed.

Limitations of $k$-anonymity are: (1) it does not hide whether a given individual is in the database [95, 105], (2) it reveals individuals' sensitive attributes [77, 81], (3) it does not protect against attacks based on background knowledge [81, 83], (4) mere knowledge of the $k$-anonymization algorithm can violate privacy [133], (5) it cannot be applied to high-dimensional data without complete loss of utility [2], and (6) special methods are required if a dataset is anonymized and published more than once [22, 119, 127].

Even though $k$-anonymity prevents neither membership disclosure, nor sensitive attribute disclosure, one may hope that $k$-anonymized datasets provide significant utility for common data-mining tasks. Unfortunately, popular metrics such as the number of generalization steps applied to quasi-identifiers, average size of quasi-identifier equivalence classes, sum of squares of class sizes, or preservation of marginals [26, 69, 81, 96] do not imply suitability of the sanitized data for data-mining computations. In Chapter 5, we follow [63, 76, 120], and measure utility by directly evaluating specific machine-learning workloads on the sanitized data.

**$\ell$-diversity.** It was recognized by Øhrn and Ohno-Machado [99] that prevention of sensitive attribute disclosure requires "diversity" of sensitive attribute values corresponding to any quasi-identifier (this is a restatement of privacy definitions from [99] using the $k$-anonymity terminology). This concept was subsequently re-invented as the $\ell$-diversity requirement. Intuitively, to prevent attribute disclosure for records in $\langle t_i \rangle$, the set of sensitive attribute values $\langle t_i \rangle[S]$ in this equivalence class should be "diverse" (which was not the case in the heart disease example shown in Table 3.2.

**Definition 5** (Entropy $\ell$-diversity [81]). *A table $T$ is Entropy $\ell$-Diverse if for each $t_i \in \mathcal{E}_{\mathcal{Q}}$,*

$$-\sum_{s \in S} p(\langle t_i \rangle, s) \log(p(\langle t_i \rangle, s)) \geq \log(\ell)$$

*where*

$$p(\langle t_i \rangle, s) = \frac{|\{t \in \langle t_i \rangle : t[\mathcal{S}] = s\}|}{|\langle t_i \rangle|},$$

*is the fraction of tuples in $\langle t_i \rangle$ with sensitive attribute value equal to $s$.*

**Definition 6** (Recursive $(c, \ell)$-diversity [81]). *Let $r_i$ denote the number of times the $i$th most frequent sensitive value appears in $\langle t_i \rangle$. Given a constant*

*$c$, $\langle t_i \rangle$ satisfies recursive $(c, \ell)$-diversity if $r_1 < c(r_\ell + r_{\ell+1} + \ldots + r_m)$. A table $T$ satisfies recursive $(c, \ell)$-diversity if for every $t_i \in \mathcal{Q}$-groups$(T)$, $\langle t_i \rangle$ satisfies recursive $(c, \ell)$-diversity. By definition, $(c, 1)$-diversity is always satisfied.*

Unfortunately, $\ell$-diversity is neither necessary, nor sufficient to prevent sensitive attribute disclosure [77]. Its approach of maximizing entropy of sensitive attribute distribution within each quasi-identifier equivalence class $\langle t_i \rangle$ relies on an implicit assumption that the distribution of sensitive attribute values in the *original* table is nearly uniform.

This assumption may or may not be true for a given database. If the distribution of sensitive attributes in the original database is skewed (generalization and suppression do not touch sensitive attributes at all, thus this distribution is always available to the adversary), then learning that some individual is in an equivalence class $\langle t_i \rangle$ where the distribution of sensitive attributes is different may reveal a lot of information, regardless of whether the latter distribution is high- or low-entropy, "diverse" or not "diverse."

For example, consider a database in which 1% of individuals have a rare form of cancer, and the quasi-identifier equivalence class $\langle t_i \rangle$ in which, say, 30% of individuals have this form of cancer (or any high percentage, as required by the diversity criterion). If the adversary's target individual $t \in \langle t_i \rangle$, then the adversary can immediately infer that his target is far more likely to have this form of cancer than a random individual in the database. In general, probabilistic sensitive attribute disclosure occurs every time when an attribute that was *not* diverse in the original database appears diverse in some quasi-identifier equivalence class in the sanitized database. On the other hand, if only 1% of individuals in the quasi-identifier equivalence class have this form of cancer, then there is no sensitive attribute disclosure, even though the class is *not* diverse.

**$t$-closeness.** Li *et al.* [77] assume that the adversary knows the distribution of sensitive attribute values over the entire table, which is a reasonable assumption because this information would have been revealed even if the quasi-identifiers had been completely suppressed. Since sanitization methods do not completely suppress the quasi-identifiers, the adversary can also determine the equivalence class $\langle t_i \rangle$ in the sanitized table to which his target individual belongs. The adversary's goal is to uncover the probability distribution of possible sensitive attribute values within $\langle t_i \rangle$.

**Definition 7** ($t$-closeness [77])**.** *An equivalence class $\langle t_i \rangle$ has $t$-closeness if the distance between the distribution of a sensitive attribute in this class $\mathcal{A}_{\mathsf{san}}(\langle t \rangle)$ and the distribution of the attribute in the whole table $\mathcal{A}_{\mathsf{base}}$ is no more than a threshold $t$. A table has $t$-closeness if all equivalence classes have $t$-closeness.*

The critical concern is how to measure the distance between distributions. Li *et al.* [77] use the *Earth Mover's distance* (EMD), which for nominal attributes is equivalent to $\mathcal{A}_{\mathsf{diff}}$. This is an additive (as opposed to multiplicative) measure, and does not translate directly into a bound on the adversary's ability to learn sensitive attributes associated with a given quasi-identifier.

|  | Zip Code | Age | Disease |
|---|---|---|---|
| 1 | 47677 | 29 | Heart Disease |
| 2 | 47602 | 22 | Heart Disease |
| 3 | 47678 | 27 | Heart Disease |
| 4 | 47905 | 43 | Flu |
| 5 | 47909 | 52 | Heart Disease |
| 6 | 47906 | 47 | Cancer |
| 7 | 47605 | 30 | Heart Disease |
| 8 | 47673 | 36 | Cancer |
| 9 | 47607 | 32 | Cancer |

Table 3.1: Original Patients Table

|  | Zip Code | Age | Disease |
|---|---|---|---|
| 1 | 476** | 20-29 | Heart Disease |
| 2 | 476** | 20-29 | Heart Disease |
| 3 | 476** | 20-29 | Heart Disease |
| 4 | 4790* | $\geq 40$ | Flu |
| 5 | 4790* | $\geq 40$ | Heart Disease |
| 6 | 4790* | $\geq 40$ | Cancer |
| 7 | 476** | 30-39 | Heart Disease |
| 8 | 476** | 30-39 | Cancer |
| 9 | 476** | 30-39 | Cancer |

Table 3.2: 3-Anonymous Version of Table 3.1

### 3.3.2 Perturbation

In the *perturbation* paradigm, statistical privacy is achieved by randomly perturbing individual data entries while preserving some global properties [4–6, 9, 35, 45, 68]. A survey can be found in [118]. Like the $k$-anonymity literature, work on making microdata private through the addition of random noise has progressed as authors have discovered definitional flaws stemming from cases not fully considered by previous work.

Agrawal and Srikant [6] used a simple distortion to hide values. Every value $x$ was replaced with $x+r$, where $r$ was a random value drawn either from a uniform or Gaussian distribution. Privacy was quantified as a *confidence level*, defined as follows:

**Definition 8** (c% confidence level [6]). *If it can be estimated with c% confidence that a value $x$ lies in the interval $[x_1, x_2]$, then the interval width $(x_2-x_1)$ defines the amount of privacy at the c% confidence level.*

For example, if we know that the distortion is uniform in the range $[-1, 1]$, then the 100% confidence interval has width 2.

Later, Agrawal and Aggarwal [4] showed that this definition failed to take into account background knowledge about the distribution of possible values for $x$ with the following example:

Let $x$ be uniformly between 0 and 1 with probability .5, and let $x$ be uniformly between 4 and 4 with probability .5. Now let us add noise $y$ drawn uniformly from the range [-1,1]. Let $z = x + y$. Then the 100% confidence level is 2, as above, and yet if $z \in [-1, 2]$ we know $x \in [0, 1]$ and if $z \in [3, 6]$ we know $x \in [4, 5]$. In either case, we know that $x$ is in an interval of width 1.

They suggest a definition of privacy based on *mutual information*, which attempts to quantify how much *more* is known about the distribution of possible values for $x$ after the perturbed value $z$ has been observed.

Later, Efimievski *et al.* [44] showed that there could still be *privacy breaches* for individual records in a database which satisfies the mutual information definition given in [4]. A privacy breach occurs when there is a substantial shift in the likelihood of some property about an individual between the *a priori* probability and the probability after seeing the sanitized database. They considered randomizers $R$ which take unperturbed elements $x$ in $V_X$ to perturbed elements $y$ in $V_Y$, and gave the following definition:

**Definition 9** ($\gamma$-amplifying [44]). *A randomization operator $R(x)$ is at most $\gamma$-amplifying for $y \in V_Y$ if*

$$\forall x_1, x_2 \in V_X : \frac{p[x_1 \to y]}{p[x_2 \to y]} \leq \gamma.$$

Operator $R(x)$ is at most $\gamma$-amplifying if it is at most $\gamma$-amplifying for all suitable $y \in V_Y$.

Intuitively, if $\gamma$ is low, then given a sanitized value $y$, most preimage values $x$ are about equally likely.

## 3.4 Perturbed Queries

In the perturbed queries scenario, a trusted database owner has access to a microdata database. A data miner repeatedly queries the database with statistical queries, such as the average, sum, or maximum value found in a particular attribute. Rather than giving the exact answers, the database owner slightly perturbs the answers in order to preserve privacy.

For interactive output perturbation, tradeoffs between privacy and utility are relatively well-understood: there are known fundamental limits on privacy [35], as well as perturbation mechanisms that provide strong semantic notions of privacy along with well-defined utility expressed in terms of machine-learning algorithms that can be computed on the perturbed data [14].

## 3.5 Differential Privacy

The differential privacy [38,39,85] definition is motivated by the following observation. Ideally, it would be possible to achieve the privacy goal that anything that can be learned about a respondent from a statistical database could also be learned without access to that database. This is analogous to the requirement of semantic security for cryptosystems 2.1.1. Unfortunately, it is impossible for a statistical database to achieve this security property while still providing any sort of useful information. The following example, taken from [39] illustrates this impossibility:

Suppose that a statistical database provides the average height for different population subgroups, and suppose that an attacker has the auxiliary information that "Terry Gross is two inches shorter than the average Lithuanian woman." Then access to the database along with this auxiliary information reveals Terry Gross' height, whereas the auxiliary information by itself does not.

The key observation is that Terry Gross is not required to be in the database in order for this disclosure to occur. Thus, she does not increase her risk of disclosure by participating in the database, nor does she decrease her risk of disclosure by refusing to participate.

Since preventing disclosure against adversaries with arbitrary auxiliary information is impossible, the differential privacy notion instead attempts to ensure that individuals take on very little risk when they agree to be part of a database. It does this by requiring that any function of the database is nearly unaffected by the insertion or removal of any single individual.

More formally, say that two databases $D_1$ and $D_2$ *differ in at most one element* if one is a subset of the other and the larger database contains one additional row. Then $\epsilon$-differential privacy is defined as follows [39]:

**Definition 10.** *A randomized function $\mathcal{K}$ gives $\epsilon$-differential privacy if for all data sets $D_1$ and $D_2$ differing on at most one element, and all $S \subseteq Range(\mathcal{K})$,*

$$\Pr[\mathcal{K}(D_1) \in S] \leq \exp(\epsilon) \times \Pr[\mathcal{K}(D_2) \in S],$$

*where the probability space in each case is over the coin flips of the mechanism $\mathcal{K}$.*

The definition of differential privacy does not distinguish between interactive and non-interactive mechanisms $\mathcal{K}$. Thus, the definition can be applied to both the sanitized database publication and the perturbed queries paradigms. Recent research has developed many mechanisms that satisfy the differential privacy definition [40–42].

Note that the privacy guarantees provided by differential privacy and secure multiparty computation are orthogonal. In the differential privacy paradigm, we are worried that the result of a computation may reveal information about database entries, so the differential privacy definition requires that no single entry can substantially influence the computation result. By contrast, secure multiparty computation assumes that the result of the computation is "private enough" and seeks to limit the disclosure of *additional* information which might be revealed during the process of obtaining the result.

## 3.6 Private Information Retrieval

A related problem is that of *private information retrieval* (PIR), in which a database contains a list of records, and a querier wishes to learn a single one of these records, but without the database owner learning which one. In the typical PIR model, the privacy of the database owner is not considered, so that a trivial (but inefficient) solution is for the database owner to transmit the entire database. In symmetric private information retrieval (SPIR) there is an additional requirement that the querier learns only the single requested record, and that the rest of the records remain private. SPIR is equivalent to the common cryptographic primitive of *oblivious transfer* 2.3.

Results in PIR include those in the *information-theoretic model*, in which there are no computational assumptions on the participants, but the database is copied among multiple non-colluding servers [25], and the *cryptographic model*, in which standard cryptographic assumptions apply [52].

*Inference control* [65, 124] can be combined with private information retrieval so that it is possible to limit the number and type of queries that are made. If an invalid query is made the query is rejected, but the database is still unable to learn what the query was, or even that it was rejected.

# Chapter 4

# Anonymous Data Collection

## 4.1 Introduction

Consider a scenario in which a data miner wishes to collect data from a large set of respondents for use in a data-mining experiment. The miner queries each respondent, who in turn transmits her response back to the miner. If the respondents are willing to truthfully answer the miner's query, then the miner is able to proceed with his experiment. In some cases, however, a respondent's willingness to answer truthfully is dependent on a guarantee that her answer will be used only in the aggregate and cannot be linked back to her. For example, the miner may be conducting a survey on illegal activities, or on sensitive medical conditions. If the miner can convince the respondent that her response will be anonymous among her peer respondents then she will participate truthfully in the survey; otherwise, she will not.

Data collected with the simple query/response protocol described above will not be anonymous, because the data miner can easily observe which respondent transmits which response. One way to achieve anonymity would be to *shuffle* the responses, so that the miner receives the responses in a random order. This is easy to do with the assistance of a trusted third-party shuffler, as the respondents can submit their responses to the shuffler, who collects all responses and then forwards them to the miner in a random order.

The goal of this research is to achieve the same security guarantees without the need for trusted parties. We give a protocol that allows mutually distrustful respondents to submit their responses to an untrusted data miner in a manner which guarantees that their responses will be received by the miner, but that the probability that the miner can link a response to a respondent is essentially no better than random guessing. Furthermore, our protocol is *collusion resistant*. Even if all malicious respondents freely share information with the malicious miner, they will be unable to learn the associations between honest respondents and their responses. This strong form of collusion resistance is important in online data collection scenarios where respondents cannot communicate directly with one another, and therefore are unable to determine whether other respondents are genuine participants, or shills set up by a malicious miner.

Our protocol consists of two parts. In the first part, respondents encrypt their responses and shuffle them under encryption, so that it will be impossible to determine which encrypted response belongs to which respondent. In the second part, the integrity of the shuffle is verified, and the respondents provide information to the data miner so that he can decrypt the responses.

In order to be practical for use in data-mining applications with large numbers of respondents, protocols for anonymous data collection need to be efficient as well as secure. In the online data collection scenario, it is especially important to limit the number of messages that must be transmitted between the data miner and the respondents. Our protocol requires that the data miner send and receive only $O(N)$ messages when there are $N$ respondents. Furthermore, our protocol does not rely on zero-knowledge proofs. The most efficient currently known zero-knowledge proofs for verifiable shuffles [59, 94] require 7 rounds of communication for each proof. Moreover, it is unclear whether these proofs preserve their properties when composed in parallel. Therefore, if multiple proofs are needed at any step of the protocol, they have to be carried sequentially, rendering the communication complexity of the protocol impractical.

## 4.2 Problem Specification

The anonymity-preserving data collection protocol takes place between a large set of mutually distrustful parties. One of these parties has a special role and is denoted the “data miner,” while the other $N$ parties have interchangeable roles and are denoted “respondents.” Each respondent $i$, $1 \leq i \leq N$ has a response $d_i$. All responses are assumed to be of identical length. The goal of the protocol is for the miner to learn the responses from each respondent, but without being able to determine which response came from which respondent. In other words, the miner should learn a random permutation of the set $\{d_1, ..., d_N\}$, but should not learn anything about that permutation.

We assume that during the protocol, all participants remain online. Each respondent has a secure communication channel with the data miner.

These are reasonable assumptions in a scenario where the respondents are using a Web interface to communicate with a server operated by the data miner. We assume that prior to the protocol execution each respondent $i$ has obtained a public encryption key pair $(x_i, y_i)$ for an IND-CCA2 encryption scheme, and a signature key pair $(u_i, v_i)$ for a secure (unforgeable) signature scheme. Each respondent and the data miner knows the public keys $y_i$ and $v_i$ for all respondents. Likewise, the data miner has a public key pair $(x_{DM}, y_{DM})$, and the public key $y_{DM}$ is known to all respondents.

In a practical implementation, the distribution of these public keys is delegated to a trusted *certification authority*, whose job is to associate individuals with their public keys. Note that this is the only assumption of trust required by the protocol. There are several businesses providing trusted certification authority functionality, so this is a reasonable (and standard) assumption.

## 4.3 Protocol Correctness

We prove the correctness of our protocol in the *malicious model* [54], where protocol participants may deviate arbitrarily from the protocol specification. In this model any participant can prevent the protocol from completing by refusing to participate, so we are unable to prove that the protocol always terminates, much less that it always terminates with correct results. Instead, we prove that if the protocol terminates, certain properties are maintained. In this section, we give three properties that together define a correct protocol for anonymity-preserving data collection in the malicious model.

### 4.3.1 Collusion resistant anonymity

In an online data collection scenario, a respondent knows nothing about her peer-respondents except their public keys. Some (or all) of the other participants may be shills controlled by the data miner who exist only to lure honest participants into a false sense of anonymity. Because dishonest respondents colluding with the data miner is a legitimate threat, we require that our protocol be *collusion resistant*. This means that even if $k$ of the $N$ respondents are corrupt and in collusion with the data miner, the data miner will be unable to determine which of the $N - k$ honest participant responses belongs to which honest participant. Of course the data miner will be able to determine whether a response comes from an honest respondent or a colluding respondent, because a colluding respondent can tell the data miner which response is hers. Note that if there is only a single honest respondent, the data miner will be able to collude with all other respondents and learn her response.

We formalize our notion of anonymity for a data collection protocol when $k$ out of the $N$ respondents are dishonest by defining an "anonymization game" which is similar to the distinguishing game given in section 2.1.1. The anonymization game is played between a challenger and an oracle, who participate in the data collection protocol together. The challenger plays the roles of the data miner and the $k$ dishonest colluding respondents, while the oracle plays the role of the honest respondents. The challenger is assumed to not know the private keys of any honest respondent. The protocol is anonymous if the challenger can win the game only with negligible probability. Prior to playing the game, the challenger may choose plaintext responses for all honest respondents and give them to the oracle, who will then participate in the anonymity-preserving data collection protocol using those responses for the honest respondents. The challenger may repeat this process polynomially many times. Then the actual game begins, and the following happens:

1. The challenger chooses two honest participants $h_\alpha$ and $h_\beta$, and two plaintext responses $m_0$ and $m_1$. He also chooses a plaintext response $d_{h_i}$ for each other honest participant $h_i$.

2. The oracle chooses a bit $b \in \{0, 1\}$ uniformly at random. Then the oracle sets $d_{h_\alpha} = m_b$ and $d_{h_\beta} = m_{\bar{b}}$.

3. The oracle participates in the anonymity protocol with the response of honest respondent $h_i$ as $d_{h_i}$. The oracle plays the role of all honest respondents. The challenger plays the role of the data miner and all dishonest respondents, and he may deviate arbitrarily from the protocol specification.

4. After observing the protocol run, the challenger guesses whether $b = 0$ or $b = 1$.

Let $D$ be a probabilistic polynomial-time challenger. Then

$$\Pr[D(m_0, m_1, h_\alpha, h_\beta, 0) = 1]$$

is the probability that $D$ outputs 1 when the bit $b = 0$, and

$$\Pr[D(m_0, m_1, h_\alpha, h_\beta, 1) = 1]$$

is the probability that $D$ outputs 1 when the bit $b = 1$. In both cases, the probability is taken over the randomness of the key generation and encryption algorithms used by the oracle. The challenger's *advantage* is equal to

$$\Pr[D(m_0, m_1, h_\alpha, h_\beta, 1) = 1] - \Pr[D(m_0, m_1, h_\alpha, h_\beta, 0) = 1].$$

**Definition 11.** *A data collection protocol is* anonymous *if, for all probabilistic polynomial-time challengers, the advantage in the anonymity game is negligible.*

Note that this definition is valid only when there are at least two honest respondents, which corresponds to our notion that it is impossible for any anonymity to exist when there is only a single honest respondent.

### 4.3.2 Integrity

Ideally, we would like to ensure that an honest data miner always receives an unaltered response from each respondent. However, this is difficult in a protocol where responses pass through every respondent during the anonymization process, as any of those respondents could be malicious (but not in collusion with the data miner) and choose to substitute some subset of the responses with other data while they are under her control. Since it seems difficult to provide authenticity guarantees on the responses while maintaining anonymity, we are satisfied to detect the occurrence of substitutions. We will say that our protocol maintains *integrity* if, at the end of protocol execution with an honest data miner, one of the following two statements is true:

1. The data miner has the correct plaintext responses for all honest respondents, or

2. The data miner is informed that some honest respondent's response has been substituted.

We are unable to make integrity claims when the data miner is dishonest, as a dishonest data miner has the power to corrupt whichever responses he wishes. However, the assumption is that the data miner is genuinely interested in learning the responses, and therefore has no such incentive.

### 4.3.3 Confidentiality

In our scenario, the respondents are taking part in a confidential survey with the data miner. The data miner should learn all plaintext responses at the end of the protocol, but a respondent should not learn any response other than her own. If the data miner is dishonest, then he can reveal the set of responses $\{d_1, ..., d_N\}$ after they have been decrypted, even though he will not know which response belongs to which respondent. We want to insure that dishonest behavior on the part of the data miner is the *only* way that the set of plaintext responses can be revealed. In other words, no coalition of dishonest respondents should be able to learn any response belonging to an honest respondent if the miner is honest.

## 4.4 Efficient Anonymous Data Collection

In this section we present our protocol for Anonymous Data Collection. We then prove that it satisfies all of the properties stated in section 4.3, and is therefore secure in the malicious model.

### 4.4.1 Protocol setup

Prior to the protocol execution, the participants must learn one another's public encryption keys and public signature keys. As is standard in cryptographic protocols, the associations between identities and keys are handled by a *certification authority*. The certification authority is a trusted party with whom every participant is assumed to have a secure communication channel. To learn the public encryption key $y_i$ and verification key $v_i$ for respondent $i$, participants query the certification authority who responds with $(y_i, v_i)$. Likewise, participants learn the public encryption key $y_{DM}$ for the data miner.

The participants must also all agree upon a canonical ordering of the respondents. This can be done, for instance, by having each respondent sign an ordering sent by the data miner, and then verifying the signatures of all other respondents.

Note that this setup needs only to be done once, and afterwards the protocol participants can perform multiple protocol executions with no need to make additional contact with the certification authority.

### 4.4.2 The protocol

Our protocol for anonymous data collection is shown in Algorithms 1–2. It consists of 5 phases: keypair generation, data submission, anonymization, verification, and decryption. These phases are described in more detail below.

#### Generation of temporary keypairs

In this phase, every respondent $i$ chooses a fresh *secondary* key pair $(w_i, z_i)$ which is distinct from the *primary* key pair $(x_i, y_i)$ that is registered with the certification authority. Each respondent $i$ then self-certifies her secondary public key $z_i$ by sending the message

$$z_i, \text{TIMESTAMP}, \text{SIG}_{u_i}\{z_i, \text{TIMESTAMP}\}$$

to the data miner, who forwards these messages to the other respondents. In this way every respondent learns the second public key $z_i$ for each other respondent $i$, which is guaranteed to be freshly chosen by $i$ because signatures are not forgeable.

### Data submission

In the data submission phase, respondents encrypt their responses first with the data miner's public key, then with all respondents' secondary public keys, and finally with all respondents' primary public keys. The encryptions are applied in the canonical ordering determined during the protocol setup. The primary key encryptions are stripped off during the anonymization phase. Because the cooperation of all respondents is necessary to remove the secondary key encryptions, every respondent will have a chance to abort the protocol before anonymity is compromised if the anonymization phase did not go according to protocol specification.

### Anonymization

In the anonymization phase, the respondents take turns shuffling the encrypted responses and removing a level of encryption. Since every level of encryption must be removed in order for the verification to pass, every respondent is ensured an opportunity to shuffle the encrypted responses. If two ciphertexts are identical, this means a dishonest participant has attempted to duplicate a response, and the protocol is aborted.

### Verification

In this phase the respondents verify that the shuffles have been done correctly. By taking advantage of the fact that each respondent knows one of the responses, this is done without the use of zero-knowledge proofs. Each respondent $i$ signs a message only if he sees his own ciphertext, $C'_i = \{C''_i\}_{z_N:z_1}$ among the set of permuted ciphertexts. These signatures are then verified by all respondents. If all of the signatures verify, an honest respondent can reason as follows:

- Every other honest respondent saw her ciphertext in the set of permuted ciphertexts.
- My own shuffle step must have included ciphertexts from all honest respondents.
- Since I performed a random shuffle and did not reveal the permutation, a dishonest data miner cannot know which honest ciphertext belongs to me.

Since the respondent knows her ciphertext is anonymous among the honest respondents' ciphertexts, she gives the data miner her secondary private key.

#### Decryption

In this phase the data miner uses the respondents' secondary private keys and his own private key to decrypt the responses. He learns the plaintext responses, but not the associations between responses and respondents.

### 4.4.3 Security arguments

In this section we argue that the security properties introduced in section 4.3 are satisfied by the protocol given in Algorithms 1–2.

- Phase 0: Secondary Keypair Generation. For $i = 1, ..., N$
  - Respondent $i$ chooses a public key pair $(w_i, z_i)$.
  - Respondent $i$ sends $z_i, \text{TIMESTAMP}, \text{SIG}_{u_i}\{z_i, \text{TIMESTAMP}\}$ to the data miner:
  - The data miner forwards it to all other respondents:
  - If any signature fails to verify, the protocol is aborted.
- Phase 1: Data submission. For $i = 1, ..., N$:
  - Respondent $i$ encrypts his data $d_i$ first with the miner's public key, and then with all respondent's secondary public keys:

    $$C_i'' = \{d_i\}_{y_{DM}} \quad , \quad C_i' = \{C_i''\}_{z_N:z_1}$$

  - Respondent $i$ stores $C_i'$ for later use, and encrypts with all respondent's primary public keys:

    $$C_i \quad = \quad \{C_i'\}_{y_N:y_1}$$

  - Respondent $i$ sends the ciphertext $C_i$ to the miner

The miner sets the initial values of $(D_1, ..., D_N)$ as $(C_1, ..., C_N)$

**Algorithm 1:** Protocol for Anonymous Data Collection (Phases 0–1)

- Phase 2: Anonymization. For $i = 1, ..., N$:
  - The miner sends $(D_1, ..., D_N)$ encrypted under the keys $y_i, ..., y_N$ to respondent $i$.
  - If any ciphertext is included more than once in $(D_1, ..., D_N)$, then respondent $i$ aborts the protocol.
  - Respondent $i$ uses her private key $x_i$ to strip off the $i$th level of encryption, and permutes the pieces. Her output is $(R_1, ..., R_N)$ where

    $$R_j \quad = \quad \mathrm{dec}_{w_i}[D_{\pi(j)}]$$

    where $\pi$ is a random permutation on $\{1, ..., N\}$.
  - Respondent $i$ sends $(R_1, ..., R_N)$ to the data miner.
  - The data miner sets $(D_1, ..., D_N) = (R_1, ..., R_N)$.
- Phase 3: Verification. At the beginning of this phase, the data miner holds $(D_1, ..., D_N)$. If the participants have behaved honestly, this should be a permutation of $(C'_1, ..., C'_N)$.
  - The data miner sends $(D_1, ..., D_N)$ to all participants.
  - Each participant $i$ verifies that $C'_i$ is included among $(D_1, ..., D_N)$. If it is, she sends $\textsc{sig}_{u_i}\{(D_1, ..., D_N)\}$ to the data miner.
  - The data miner forwards signatures $\textsc{sig}_{u_i}\{(D_1, ..., D_N)\}$, $i = 1, ..., N$ to all respondents.
  - Each respondent $i$ verifies the signatures. If they *all* verify, respondent $i$ sends his secondary private key $w_i$ to the data miner.
- Phase 4: Decryption. Using the keys $w_1, ..., w_N$ and $x_{DM}$, the data miner removes the remaining levels of encryption from $D_1, ..., D_N$, resulting in a permutation of the original responses $d_1, ..., d_N$.

**Algorithm 2:** Protocol for Anonymous Data Collection (Phases 2–4)

## Anonymity

An intuitive argument for the anonymity of the protocol is that the data miner and colluding respondents have two choices for their behavior. On the one hand, they can behave honestly, in which case they will learn the final decrypted plaintexts, but they will not learn the associations between $C'$ and $C$ ciphertexts. On the other hand, they can behave dishonestly and learn some associations between $C'$ and $C$ ciphertexts, but then the verification phase will fail and they will not learn the decryptions of the $C'$ ciphertexts.

Our proof is in two parts. First, we show that when honest respondents receive ciphertexts for decryption in phase 2, then either there is exactly one copy of the correct ciphertext for each honest participant, or the deviation from the protocol is detected and the protocol is aborted before the challenger is able to win the verification game. Second, we show that a challenger who can win the anonymity game while maintaining the above property can also win the distinguishing game, which is a contradiction because the underlying encryption scheme is IND-CCA2.

*Part 1:* Suppose that during step $h_i$ of phase 2, honest respondent $h_i$ receives ciphertexts $(D_1, ..., D_N)$, but that there is some honest respondent $h_p$ for which the ciphertext $\{C'_{h_p}\}_{y_N:y_{h_i}}$ appears either more than once or not at all. If any ciphertext appears more than once, this is detected by honest respondent $h_i$ and the protocol is aborted before the challenger learns the secondary private keys.

Now we wish to show that if an honest ciphertext is dishonestly replaced so that it does not appear in step $h_i$ of phase 2, then the verification in phase 3 will fail. Suppose that honest participant $h_i$ does not receive $\{C'_{h_p}\}_{y_N:y_{h_i}}$ as part of the set $(D_1, ..., D_N)$.

In this case it is infeasible for the challenger to learn $\{C'_{h_p}\}_{y_N:y_{h_i-1}}$ because they did not give $\{C'_{h_p}\}_{y_N:y_{h_i}}$ to participant $h_i$. However, they must learn $C'_{h_p}$ to satisfy honest participant $h_p$ in the verification step. This is infeasible without the private key $x_{h_i}$. Therefore verification fails and the protocol is aborted before the challenger learns the secondary private keys.

Now we must show that the challenger cannot win the anonymization game when he does not learn the secondary private keys. It is possible that by duplicating or substituting ciphertexts of honest respondents, the challenger can learn the partially-decrypted ciphertexts $C'_{h_\alpha}$ and $C'_{h_\beta}$ for the honest respondents $h_\alpha$ and $h_\beta$ he has chosen in the game. For example, the challenger could substitute known information for all encrypted responses except $C_{h_\alpha}$. Then after the decryption phase, he would know $C'_{h_\alpha}$. However, even if the challenger learns $C'_{h_\alpha}$ and $C'_{h_\beta}$, each of these ciphertexts is encrypted with the key $z_{h_\alpha}$ (as well as the keys of all other honest participants) which is unknown to the challenger. Therefore determining which decrypts to $m_0$ and which decrypts to $m_1$ is *exactly equivalent* to the distinguishing game, and cannot be done due to the assumption that the encryption scheme is IND-CCA2.

*Part 2:* Now suppose that the challenger honestly handles all ciphertexts belonging to honest participants. Suppose also that there is a probabilistic polynomial time algorithm $D$ that allows this challenger to win the anonymization game with non-negligible probability. We will show how to use $D$ as a subroutine to probabilistic polynomial-time algorithm $A$ that wins the distinguishing game with non-negligible probability. Because of the assumption that the underlying encryption scheme is IND-CCA2, this is a contradiction, and we will conclude that no such $D$ exists. Let the set of $k$ honest respondents in the anonymization protocol be $H = \{h_1, ..., h_k\}$. Let $D$ be an algorithm that allows the challenger to win the anonymization game with non-negligible probability. Then there exist honest participants $h_\alpha$ and $h_\beta$ such that for some polynomial $f$, and for sufficiently large values of $\rho$ ($\rho$ is the security parameter [55]),

$$\begin{aligned} & \Pr[D(m_0, m_1, h_\alpha, h_\beta, 1) = 1] \\ - & \Pr[D(m_0, m_1, h_\alpha, h_\beta, 0) = 1] \\ > & \frac{1}{f(\rho)}. \end{aligned}$$

To apply $D$, $A$ must simulate the oracle in the anonymization game to reproduce the view of the challenger. We show how $A$ is able to do this.

Algorithm $A$ begins by asking the distinguishing game oracle to generate a keypair $(x_o, y_o)$. $A$ will use $y_o$ for $y_{h_k}$, the public key belonging to the last honest respondent $h_k$. $A$ will be able to simulate all messages from honest respondents despite not knowing $x_o$ or $y_o$.

$A$ applies $D$ to learn its choices in step 1 of the anonymization game. $A$ therefore learns the following:

- Two honest participants $h_\alpha$ and $h_\beta$, and two plaintext responses $m_0$ and $m_1$
- A plaintext response $d_{h_i}$ for each other honest respondent $h_i$

Then, for each honest respondent $h_i$, $A$ chooses

- $(x_{h_i}, y_{h_i})$, a primary keypair (not chosen for respondent $h_k$),
- $(w_{h_i}, z_{h_i})$, a secondary keypair, and
- $(u_{h_i}, v_{h_i})$, a signature keypair

$A$ selects plaintexts $m_0$ and $m_1$ to be the plaintext responses for $h_\alpha$ and $h_\beta$.

$A$ is now ready to play the role of the oracle in the anonymization game by simulating the messages of honest participants in the protocol execution. For each phase of the protocol, we will explain how $A$ is able to reproduce the messages sent in that phase.

*Phase 0*: $A$ has all the necessary keys to reproduce this phase exactly.

*Phase 1*: For all honest respondents $h_i$ other than $h_\alpha$ and $h_\beta$, $A$ encrypts response $d_{h_i}$ using the appropriate public keys (and the encryption oracle to encrypt with $y_{h_k}$) which results in the ciphertext $C_i$. $A$ then encrypts $m_0$ and $m_1$ in a special way. First, $A$ encrypts using the secondary public keys:

$$\begin{aligned} m_0' &= \{m_0\}_{z_N:z_1}, \text{ and} \\ m_1' &= \{m_1\}_{z_N:z_1}. \end{aligned}$$

Next, $A$ encrypts with all keys that come after $h_k$ in the sequence:

$$\begin{aligned} M_0 &= \{m_0'\}_{y_N:y_{h_k-1}}, \text{ and} \\ M_1 &= \{m_1'\}_{y_N:y_{h_k-1}}. \end{aligned}$$

Next, $A$ gives $M_0$ and $M_1$ to the distinguishing game oracle, getting back:

$$\begin{aligned} M_b &= \{m_b'\}_{y_N:y_{h_k}}, \text{ and} \\ M_{\bar{b}} &= \{m_{\bar{b}}'\}_{y_N:y_{h_k}}. \end{aligned}$$

Finally, $A$ encrypts $M_b$ and $M_{\bar{b}}$ with the remaining public keys to get the final encryptions:

$$\begin{aligned} C_{h_\alpha} &= \{m_b'\}_{y_N:y_1}, \text{ and} \\ C_{h_\beta} &= \{m_{\bar{b}}'\}_{y_N:y_1}. \end{aligned}$$

*Phase 2*: For all honest participant rounds $h_i$ of phase 2 other than round $h_k$, $A$ has the key $x_{h_i}$ necessary to decrypt the ciphertexts $(D_1, ..., D_N)$ provided by the challenger (who is playing the role of the miner). $A$ permutes the resulting decryptions and sends them to the challenger.

Round $h_k$ is more difficult, because $A$ does not know the key $x_{h_k}$. Here we must use our assumption that every honest participant's ciphertext appears exactly once in $(D_1, ..., D_N)$. In particular, the ciphertexts

$$\begin{aligned} M_b &= \{m_b'\}_{y_N:y_{h_k}}, \text{ and} \\ M_{\bar{b}} &= \{m_{\bar{b}}'\}_{y_N:y_{h_k}}. \end{aligned}$$

appear exactly once. For all ciphertexts $D_j$ in $(D_1, ..., D_N)$ other than $M_b$ and $M_{\bar{b}}$, $A$ may use the decryption oracle provided by the distinguishing game to obtain $\text{dec}_{x_{h_k}}(D_j)$.

However, $A$ is not allowed to use the decryption oracle on $M_b$ or $M_{\bar{b}}$. Instead, when $A$ sees the ciphertexts $\{m'_b\}_{y_N:y_{h_k}}$ and $\{m'_{\bar{b}}\}_{y_N:y_{h_k}}$, he can either simulate decryption as

$$\begin{aligned} &\{m'_0\}_{y_N:y_{h_k-1}} \quad \text{and} \quad \{m'_1\}_{y_N:y_{h_k-1}} \text{ or} \\ &\{m'_1\}_{y_N:y_{h_k-1}} \quad \text{and} \quad \{m'_0\}_{y_N:y_{h_k-1}}, \end{aligned}$$

which are the values $M_0$ and $M_1$ known to $A$. Because $A$ sends the challenger a random permutation of the decryptions, the challenger cannot distinguish between $A$ making the correct choice and permuting, or $A$ making the incorrect choice and permuting. Thus either choice will suffice, and $A$ may randomly choose one or the other.

*Phases 3 and 4*: $A$ has all the necessary keys to reproduce these phases exactly.

Now $A$ simulates the view of the challenger and applies $D$ to the view. If $D$ outputs 1, then $A$ outputs 1, and if $D$ outputs 0, $A$ outputs 0.

We will now analyze the probability of $A$ outputting 1 if the distinguishing oracle chose $b = 0$ and if the distinguishing oracle chose $b = 1$. If $b = 0$, then the view of the challenger is $(m_0, m_1, h_\alpha, h_\beta, 0)$. If $b = 1$, then the view of the challenger is $(m_0, m_1, h_\alpha, h_\beta, 1)$. Let

$$p_0 = \Pr[D(m_0, m_1, h_\alpha, h_\beta, 0) = 1],$$

$$p_1 = \Pr[D(m_0, m_1, h_\alpha, h_\beta, 1) = 1].$$

Based on our assumption that $D$ wins the anonymity game, we have that $p_1 - p_0 > \frac{1}{f(\rho)}$. Now we make a simple substitution,

$$\begin{aligned} & \Pr[A(m_0, m_1, 1) = 1] - \Pr[A(m_0, m_1, 0) = 1] \\ = \ & p_1 - p_0 \\ > \ & \frac{1}{f(\rho)}. \end{aligned}$$

We conclude that $A$ can win the distinguishing game with non-negligible probability, which contradicts the IND-CCA2 property of the underlying encryption scheme.

**Integrity**

The honest respondents verify the presence of their $C_i'$ ciphertext in phase 3 of the protocol; all ciphertexts must be present for any decryption to take place. When the secondary private keys are handed over to the data miner for decryption, he can easily confirm whether the private key $w_i$ actually corresponds to the public key $z_i$ by encrypting data with $z_i$ and determining whether it decrypts with $w_i$. Then the data miner uses verified keys to decrypt verified ciphertexts, and as a result learns the correct plaintext responses.

If the verification in phase 3 fails, then the data miner has not received $\textsc{sig}_{u_i}(D_1, ..., D_N)$ from some honest participant $h_i$. In this case the data miner learns that participant $h_i$'s response has been substituted. Thus our two-part definition of integrity given in section 4.3.2 is satisfied.

### Confidentiality

The inner-most level of encryption on each response is with $y_{DM}$. Therefore only the data miner can perform the final decryption and learn the responses.

## 4.5 Efficiency

Our protocol is efficient in terms of communication and computational complexity. In this section we will quantify the complexity of the protocol.

### Computation

Operations requiring computation include key pair generation, encryption, decryption, signing, and signature verification. In phase 0, each respondent generates 1 key pair, signs 1 message, and performs $N$ signature verifications. In phase 1, each respondent performs $2N + 1$ encryptions. In phase 2, each respondent performs $N$ decryptions. In phase 3, each respondent performs 1 signature and $N$ signature verifications. In phase 4, the data miner performs $N^2 + N$ decryptions. We conclude that the total computational complexity is $O(N^2)$. Note, however, that the computational complexity for each individual respondent is only $O(N)$. This is advantageous because the respondents are more likely to be computationally bounded than the data miner.

**Number of Communication Rounds**

Phase 0 is parallelizable and requires 2 rounds. Phase 1 is parallelizable and requires only 1 round. Phase 2 cannot be parallelized and requires $2N$ rounds. Phase 3 can be parallelized and requires 4 rounds. Phase 4 does not involve communication. We conclude that the protocol requires $O(N)$ communication rounds.

**Total Communication**

Let us assume that the size of a response is $S$ bits, the size of a key is $T$ bits, and that the size of a signature is $Q$ bits. Phase 0 requires the transmission of $N^2$ signatures and keys, for a total of $(Q+T)N^2$ bits. In phase 1, $N$ $S$-bit ciphertexts are transmitted. In each iteration of phase 2 , $2SN$ bits are transmitted, for a total of $2SN^2$ bits. Phase 3 transmits $SN^2$ bits for the broadcast of $(D_1, ..., D_N)$, $(Q+T)N$ bits to transmit signatures and keys to the miner, and $QN^2$ bits to broadcast the signatures from the miner back to the respondents. We conclude that the protocol transmits $O((Q+S+T)N^2)$ bits. Since $Q$ and $T$ are constant parameters of the cryptosystem, we can simplify this to $O(SN^2)$ bits.

## 4.6 An Attack on the YZW Protocol

In this section, we show an attack against the data collection protocol of Yang *et al.* [129, section 5.2]. We will refer to this protocol as the YZW protocol. This protocol is intended to be collusion resistant so that $t-1$

dishonest leaders (out of $t$ total leaders), in collusion with a dishonest data miner, cannot learn any associations between honest respondents and their responses. Due to a subtle flaw in the use of zero-knowledge proofs, it is actually possible for a single dishonest leader to collude with the data miner and learn *all* associations.

### 4.6.1 Description of the YZW protocol

Unlike our protocol, which is compatible with any IND-CCA2 cryptosystem, the YZW protocol relies on the ElGamal encryption scheme. In ElGamal, the public key is $(p, g, g^x)$ and the private key is $x$, where $p$ is a large random prime, $g$ is the generator of the multiplicative group of integers modulo $p$, and $x$ is a random integer such that $1 \leq x \leq p-2$. A ciphertext of message $m$ is a pair $(g^k \mod p, m \cdot (g^x)^k \mod p)$, where $k$ is a random integer such that $1 \leq k \leq p-2$.

An important property of ElGamal ciphertexts is that they can be easily *rerandomized* without access to the private key, *i.e.*, given a ciphertext $(C^{(1)}, C^{(2)})$, it is easy to produce, without decrypting, another ciphertext $(C^{(1)} \cdot g^{k'} \mod p, C^{(2)} \cdot (g^x)^{k'} \mod p)$ which encrypts the same plaintext. Moreover, given two ciphertexts, it is not feasible to determine whether they encrypt the same plaintext.

The protocol also makes use of the following three zero-knowledge proofs, which are intended to prevent any of the protocol participants from deviating from the protocol:

- $\mathsf{PoK}(C)$, where $C$ is an ElGamal ciphertext. This is a proof of knowledge of the plaintext of $C$.

- $\mathsf{PoR}((C_1, ..., C_N), (C'_1, ..., C'_N))$, where all $C_i$ and $C'_i$ are ElGamal ciphertexts. This is a proof that $(C'_1, ..., C'_N)$ is a permuted rerandomization of $(C_1, ..., C_N)$.

- $\mathsf{PoD}(q, C^{(2)}, y)$, where $C^{(2)}$ is the second component of an ElGamal ciphertext and $y = g^x \mod p$ is a public key. This is a proof that $q = (C^{(2)})^x$, where $x$ is the private key corresponding to $y$.

We restate the protocol below. There are $N$ respondents, of whom $t$ are designated as "leaders." Each leader $i$ has an ElGamal key pair, in which $x_i$ is the private key, and the public key includes $y_i = g^{x_i}$. The public key is known to all respondents, while the private key $x_i$ is known only to leader $i$. Let

$$y = \prod_{y=1}^{t} y_i$$

be the product of all public $y_i$ values. The protocol consists of three phases:

- Phase 1: $N$-round data submission.

  For $i = 1, ..., N$

  - Respondent $i$ encrypts his data $d_i$ using the public key $y$ to produce the ciphertext $C_i$:

$$C_i \overset{\text{def}}{=} (C_i^{(1)}, C_i^{(2)}) = (y^{r_i} d_i, g^{r_i}),$$

where $r_i$ is picked uniformly at random.

- Respondent $i$ produces a proof $z_i = \mathsf{PoK}(C_i)$, proving that he knows the plaintext of $C_i$.
- Respondent $i$ sends $C_i$ and $z_i$ to the miner, who forwards $(C_i, z_i)$ to all other respondents.
- Each respondent verifies the proof sent by respondent $i$ and if it is missing or invalid, the protocol is aborted.
- The data miner sets the initial values of $(D_1, ..., D_N)$ to be $(C_1, ..., C_N)$.

• Phase 2: $t$-round anonymization.
For $i = 1, ..., t$

- The miner sends $(D_1, ..., D_N)$ to leader $i$.
- Leader $i$ rerandomizes and permutes the data, so that $(R_1, ..., R_N)$ is a permuted rerandomization of $(D_1, ..., D_N)$.
- Leader $i$ generates a proof

$$w_i = \mathsf{PoR}((D_1, ..., D_N), (R_1, ..., R_N)).$$

- Leader $i$ sends $(R_1, ..., R_N)$ and $w_i$ to the miner, who forwards them to all other respondents.
- Each respondent verifies the proof sent by leader $i$ and if it is missing or invalid, the protocol is aborted.

    - The miner sets the new values of $(D_1, ..., D_N) = (R_1, ..., R_N)$.

- Phase 3: Decryption
    - The miner sends $(D_1, ..., D_N)$ to all leaders.
    - Each leader $i$ computes partial decryptions: for $j = 1, ..., N$,
$$p_{j,i} = (D_j^{(2)})^{x_i}.$$
    - Each leader $i$ computes a proof
$$v_{j,i} = \mathsf{PoD}(p_{j,i}, D_j^{(2)}, y_i).$$
    - Each leader $i$ sends $p_{j,i}$ and $v_{j,i}$ to the miner, for $j = 1, ..., N$. The miner forwards them to the other participants.
    - Each participant verifies the proof sent by leader $i$, and if it is missing or invalid the protocol is aborted.
    - The miner computes the final decryptions: for $j = 1, ..., N$
$$d'_j = D_j^{(1)} / \prod_{i=1}^{t} p_{j,i}.$$

### 4.6.2 Attacking the YZW protocol

The flaw in the YZW protocol is that the zero-knowledge proof of permuted rerandomization $\mathsf{PoR}$ is *not* a proof of knowledge. It guarantees that the plaintexts of two ciphertext sets are the same, but this is not enough for anonymity. As long as $(C'_1, ..., C'_N)$ is *some* permuted rerandomization of

$(C_1, ..., C_N)$, then an attacker can provide the required proof even if he does *not* know the actual permutation.

In this section, we show how this can be exploited by a malicious data miner who colludes with the last leader and substitutes original ciphertexts (for which associations with respondents are known) for the honestly permuted ciphertexts. This enables them to pass all proofs required by the protocol, and then learn all associations between respondents and responses. Collusion resistance thus fails completely: it is sufficient for the data miner to corrupt the last leader in order to completely break security of the protocol.

1. All participants behave honestly until the beginning of the $t$th round of phase 2, when it is leader $t$'s turn to rerandomize and permute the data.

2. At the beginning of the $t$th round, the data miner sends leader $t$ both the current values of $(D_1, ..., D_N)$ and the *original* values of $(C_1, ..., C_N)$ exactly as they were collected from the respondents during phase 1. Leader $t$ produces $(R_1, ..., R_N)$ by rerandomizing and applying a permutation $\pi_t$ to $(C_1, ..., C_N)$, *not* $(D_1, ..., D_N)$ as the protocol specifies. Because $(R_1, ..., R_N)$ is also a permuted rerandomization of $(D_1, ..., D_N)$, leader $t$ is able to produce the proof

$$\mathsf{PoR}((D_1, ..., D_N), (R_1, ..., R_N)),$$

even though he does not know the permutation to produce $(R_1, ..., R_N)$ from $(D_1, ..., D_N)$.

3. The proof $\mathsf{PoR}((D_1, ..., D_N), (R_1, ..., R_N))$ from leader $t$ is verified by all other leaders, who then decrypt in phase 3. The data miner learns all plaintext responses permuted only by $\pi_t$

4. Leader $t$ tells $\pi_t$ to the data miner, who is then able to associate responses to respondents.

It may appear that this attack is caused simply by an imprecise description of $\mathsf{PoR}$ given in [129], and that any actual implementation of the $\mathsf{PoR}$ proof would not allow a party to pass the proof $\mathsf{PoR}((D_1, ..., D_N), (R_1, ..., R_N))$ without knowing the permutation. Unfortunately, the implementation suggested in [129] (and originally proposed in [57]) allows precisely this attack.

The proof in [57, 129] relies on the multiplicative homomorphism property of ElGamal. If $(C_1^{(1)}, C_1^{(2)})$ and $(C_2^{(1)}, C_2^{(2)})$ are ElGamal encryptions of plaintexts $P_1$ and $P_2$, then

$$(C_1^{(1)} C_2^{(1)}, C_1^{(2)} C_2^{(2)})$$

is an ElGamal encryption of $P_1 P_2$. In order to prove that two sets of ciphertexts $(C_1, ..., C_N)$ and $(R_1, ..., R_N)$ decrypt to the same set of plaintexts, participants are asked to prove that the products of the ciphertexts

$$(\prod_{i=1}^{N} C_i^{(1)}, \prod_{i=1}^{N} C_i^{(2)}) \quad \text{and}$$
$$(\prod_{i=1}^{N} R_i^{(1)}, \prod_{i=1}^{N} R_i^{(2)}),$$

decrypt to the same values (which are products of the original plaintexts). If plaintexts are chosen in such a way that it is difficult to find a set of plaintexts with the same product as the original, then equality of the products of plaintexts implies equality of the plaintexts themselves.

The reason the protocol of [129] fails is that the above statement about equality of products is true, and a malicious leader can prove it, even if he does not know the permutation. In [123], Wikström presents a clever attack that effectively allows the $k$th leader to present a convincing zero-knowledge proof that his output $(R_1, ..., R_N)$ is a permuted rerandomization of the output from the $k-1$st leader, when in fact $(R_1, ..., R_N)$ is a permuted rerandomization of the original ciphertexts in their original order. (In [123], the attack is described in the context of mix networks, but it also works in the anonymous data collection setting with very slight modifications.)

### 4.6.3 Fixing the YZW protocol

The YZW protocol can potentially be fixed by substituting zero-knowledge proofs of knowledge for the incorrect proofs of [57]. Research on so called verifiable shuffles [59, 94] has led to zero-knowledge proofs in which a participant in an ElGamal rerandomization protocol proves not only that the output ciphertexts decrypt to the same set of plaintexts as the input ciphertexts, but also that the prover knows the the permutation from input ciphertexts to output ciphertexts.

Requiring this additional proof of knowledge makes the attack described above impossible, since a malicious last leader will no longer be able to rerandomize original input ciphertexts instead of the ciphertexts provided by the previous leader and pass the proof of knowledge. Fixing the YZW protocol in this way, however, still requires $O(N^2)$ communication rounds and the use of expensive zero-knowledge proofs, where $N$ is the number of participants. It is not clear whether the proofs of [59, 94] can be carried out in parallel (in general, zero-knowledge proofs do not preserve their properties under concurrent composition), and executing them sequentially results in a protocol with impractical communication complexity. By contrast, our protocol achieves the same security guarantees with $O(N)$ communication rounds and without any zero-knowledge proofs.

## 4.7 Conclusions

We have presented an efficient protocol for anonymity-preserving data collection that does not rely on zero-knowledge proofs to be secure in the malicious model. We have provided anonymity by having the respondents function as mix-servers which shuffle the set of responses. The critical insight of our research is that by taking advantage of the fact that each respondent/mix-server knows her own response, we can confirm the validity of the shuffles without using zero-knowledge proofs. In a traditional mix-net scenario, the mix-servers and the data providers are distinct entities, so validity confirmation of this type is not possible. It is our hope that the data-mining community will find our protocol useful when collecting sensitive data from respondents.

# Chapter 5

# Tradeoffs in Sanitized Data Publishing

## 5.1 Introduction

Microdata records contain information about specific individuals. Examples include medical records used in public-health research, individual transactions or preferences released to support the development of new data-mining algorithms, and records published to satisfy legal requirements.

In contrast to statistical databases and randomized response methods, the records in question contain actual, unperturbed data associated with individuals. Some of the attributes may be sensitive, *e.g.*, health-related attributes in medical records. Therefore, identifying attributes such as names and Social Security numbers are typically removed from microdata records prior to release. The published records may still contain "quasi-identifiers," *e.g.*, demographic attributes such as ZIP code, age, or sex. Even though the quasi-identifier attributes do not directly reveal a person's identity, they may appear together with the identity in another public database, or it may be easy to reconstruct their values for any given individual. Microdata records may also contain "neutral" attributes which are neither quasi-identifying, nor sensitive.

The association of quasi-identifiers with sensitive attributes in public records has long been recognized as a privacy risk [73, 110]. This type of privacy breach is known as *sensitive attribute disclosure*, and is different from *membership disclosure*, *i.e.*, learning whether a certain individual is included in the database [38, 95, 105].

It is very easy to prevent sensitive attribute disclosure by simply not publishing quasi-identifiers and sensitive attributes together. *Trivial sanitization* that removes either all quasi-identifiers, or all sensitive attributes in each data release provides the maximum privacy possible against an adversary whose knowledge about specific individuals is limited to their quasi-identifiers (this adversary is very weak, yet standard in the microdata sanitization literature [26, 81, 112]).

There is large body of research on techniques such as $k$-anonymity and $\ell$-diversity that apply domain-specific generalization and suppression to quasi-identifier attributes and then publish them together with unmodified sensitive attributes. In this chapter, we ask a basic question: **what benefit do these algorithms provide over trivial sanitization?** The only reason to publish generalized quasi-identifiers and sensitive attributes together is to support data-mining tasks that consider both types of attributes in the sanitized database. Our goal in this chapter is to evaluate the tradeoff between this incremental gain in data-mining utility and the degradation in privacy caused by publishing quasi-identifiers together with sensitive attributes.

**Our contributions.** First, we give a *semantic* definition of sensitive attribute disclosure. It captures the gain in the adversary's knowledge due to his observations of the sanitized dataset. This definition is somewhat similar to privacy definitions used in random-perturbation databases [44], but is adapted to the generalization and suppression framework.

Second, we give a methodology for measuring the tradeoff between the loss of privacy and the gain of utility. Privacy loss is the increase in the adversary's ability to learn sensitive attributes corresponding to a given identity. Utility gain is the increase in the accuracy of machine-learning tasks evaluated on the sanitized dataset. The baseline for both is the *trivially sanitized* dataset, which simply omits either all quasi-identifiers, or all sensitive attributes, thus providing maximum privacy and minimum utility.

Third, we evaluate our methodology on the same datasets from the UCI machine learning repository as used in previous research on sanitized microdata utility [76,77,81]. We show that non-trivial generalization and suppression either results in large privacy breaches, or provides little incremental utility vs. a trivially sanitized dataset. Therefore, even if the adversary's knowledge is limited to quasi-identifiers, the data-mining utility must be destroyed to achieve only marginal privacy. To protect against an adversary with auxiliary knowledge, the loss of utility must be even greater.

## 5.2 Definitions and Notation

Let $T = \{t_1, \ldots t_n\}$ be a data table. Each $t_i$ is a tuple of attribute values representing some individual's record. Let $\mathcal{A} = \{a_1, \ldots a_m\}$ be the set of attributes; $t[a_i]$ denotes the value of attribute $a_i$ for tuple $t$. We use the following notation for subsets of attributes and tuples. If $\mathcal{C} = \{c_1, c_2, \ldots c_p\} \subseteq A$, then $t[C]$ denotes $(t[c_1], \ldots t[c_p])$. If $U = \{u_1, u_2, \ldots u_p\} \subseteq T$, then $U[a]$ denotes $(u_1[a], \ldots u_p[a])$.

Let $\mathcal{S} \in \mathcal{A}$ be the *sensitive attribute*. This is an attribute whose value the adversary should not be able to associate with an individual (*e.g.*, medical information). Let $S = \{s_1, \ldots s_l\}$ be the set of possible attribute values for the sensitive attribute $\mathcal{S}$. All of the concepts in this chapter are easily explained in the single sensitive attribute setting, but can also be generalized to multiple sensitive attributes.

Let $\mathcal{Q} \in \mathcal{A} \backslash \mathcal{S}$ be the *quasi-identifier*, *i.e.*, the set of non-sensitive (*e.g.*, demographic) attributes whose values may be known to the adversary for a given individual.

Two tuples $t_i$ and $t_j$ are $\mathcal{Q}$-equivalent (denoted $t_i \overset{\mathcal{Q}}{\equiv} t_j$) if $t_i[\mathcal{Q}] = t_j[\mathcal{Q}]$. This equivalence relation partitions $T$ into quasi-identifier equivalence classes, denoted as $\langle t_j \rangle$, where $t_i \in \langle t_j \rangle$ iff $t_i \overset{\mathcal{Q}}{\equiv} t_j$. Let $\mathcal{E}_{\mathcal{Q}} \subseteq T$ be a set of representative records for each equivalence class imposed by $\overset{\mathcal{Q}}{\equiv}$.

We make the standard assumption that the adversary knows only the quasi-identifiers [26,74,112,128]. This weak adversary model makes our results *stronger* because if privacy fails against the weak adversary, it will also fail against adversaries who have additional knowledge [81, 83].

$T$ may also contain attributes in $\mathcal{A} \setminus (\mathcal{Q} \cup \mathcal{S})$, which are neither sensitive, nor quasi-identifying. For example, a user may wish to construct a classifier that predicts the values of these "neutral" attributes (*e.g.*, length of hospital stay) on the basis of both quasi-identifiers (*e.g.*, age) and sensitive attributes (*e.g.*, diagnosis).

Consider a subset of tuples $U = \{u_1, u_2, \ldots u_p\} \subseteq T$, and the distribution of sensitive attribute values within $U$. For any sensitive attribute value $s$, denote by $U_s$ the set $\{u \in U \mid u[\mathcal{S}] = s\}$ of tuples in $U$ whose sensitive attribute value is equal to $s$, and denote by $p(U, s)$ the corresponding fraction of tuples in $U$, computed as $\frac{|U_s|}{|U|}$. The notation $p(U, s)$ can be understood as "the probability that a randomly chosen member of $U$ has sensitive attribute value $s$."

We assume that whenever an adversary is provided with a sanitized table $T'$, the record rows appear in a random order to prevent "unsorted matching attacks" [112].

## 5.3 Sensitive Attribute Disclosure

Sensitive attribute disclosure occurs when the adversary learns information about an individual's sensitive attribute(s). To obtain a meaningful definition of data privacy, it is necessary to quantify the knowledge about sensitive attributes that the adversary gains from observing the sanitized database. We call our definitions *semantic* because they capture this shift in the adversary's knowledge. The need for semantic definitions of privacy is well-understood for random-perturbation databases (*e.g.*, [44]). By contrast, research on microdata privacy has focused on purely *syntactic* privacy definitions such as $k$-anonymity and $\ell$-diversity (surveyed in Section 3.3.1), which only consider the distribution of attribute values in the sanitized database, without directly measuring what the adversary may learn.

### 5.3.1 Attack model

We use the standard model from the literature [26, 81]. The adversary is given a sanitized table $T'$ generated from an original table $T$, and the quasi-identifier $t[\mathcal{Q}]$ for some target individual $t$ known to be in the table $T$ (*i.e.*, we are not considering membership disclosure). We re-emphasize that giving the adversary *more* background knowledge will result in even worse disclosure than we demonstrate.

To keep the sanitized database "truthful" [107, 112], generalization and suppression are applied only to quasi-identifiers, with sensitive attributes left intact. Therefore, the most "private" sanitized table possible with this approach is the trivial sanitization in which all $\mathcal{Q}$ are suppressed. Equally effective is the trivial sanitization in which all $\mathcal{S}$ are suppressed (and released in a separate, unlinked table).

The adversary's *baseline knowledge* $\mathcal{A}_{\mathsf{base}}$ is the minimum information about sensitive attributes that he can learn after any sanitization, including trivial sanitization which releases quasi-identifiers and sensitive attributes separately. $\mathcal{A}_{\mathsf{base}}$ is the distribution of sensitive attributes in the original table, which is revealed by *any* generalization and suppression algorithm because sensitive attributes are left untouched to keep them "truthful." We are concerned about privacy leaks *in excess of* this baseline knowledge; for example, if 90% of the individuals in $T$ have cancer, then it should *not* be considered an attribute disclosure if the adversary concludes that $t$ has cancer with probability 90%, since this baseline distribution is always revealed to the adversary. We formally define $\mathcal{A}_{\mathsf{base}}$ as the vector of probabilities representing the distribution of sensitive attribute values in the entire table $T$: $\mathcal{A}_{\mathsf{base}} = \langle p(T, s_1), p(T, s_2), \ldots, p(T, s_l)\rangle$.

The adversary's *posterior knowledge* $\mathcal{A}_{\mathsf{san}}$ is what he learns from the sanitized table $T'$ about the sensitive attributes of his target individual $t \in T$. Unlike $\mathcal{A}_{\mathsf{base}}$, $\mathcal{A}_{\mathsf{san}}$ takes quasi-identifiers into account, because the records in $T'$ contain a mixture of generalized and suppressed quasi-identifiers. Because the generalization hierarchy on quasi-identifiers is required to be totally ordered [26], the adversary can uniquely identify the quasi-identifier equivalence class $\langle t\rangle$ containing the sanitized record of $t$ in $T'$. $\mathcal{A}_{\mathsf{san}}$ is the distribution of sensitive attribute values within this class $\langle t\rangle$: $\mathcal{A}_{\mathsf{san}}(\langle t\rangle) = \langle p(\langle t\rangle, s_1), p(\langle t\rangle, s_2), \ldots, p(\langle t\rangle, s_l)\rangle$.

*Sensitive attribute disclosure* is the difference between the adversary's posterior knowledge $\mathcal{A}_{\mathsf{san}}$ and his baseline knowledge $\mathcal{A}_{\mathsf{base}}$. It can measured additively or multiplicatively.

$$\mathcal{A}_{\mathsf{diff}}(\langle t \rangle) = \tfrac{1}{2}\textstyle\sum_{i=1}^{l} |p(T, s_i) - p(\langle t \rangle, s_i)|$$
$$\mathcal{A}_{\mathsf{quot}}(\langle t \rangle) = \left|\log \tfrac{p(\langle t \rangle, s)}{p(T,s)}\right|$$

Informally, it captures how much *more* the adversary learns by observing sanitized quasi-identifiers than he would have learned from a "maximally private" database where sensitive attributes are separated from the quasi-identifiers.

### 5.3.2 Semantic privacy

To capture the incremental gain in the adversary's knowledge caused by the sanitized table $T'$, we first consider his baseline knowledge $\mathcal{A}_{\mathsf{base}}$ as defined above. Recall that it consists of the distribution of sensitive attributes in the table $T^*$, where all quasi-identifiers have been suppressed (any sanitization that does not touch sensitive attributes necessarily reveals $T^*$). Furthermore, the adversary knows $t[\mathcal{Q}]$ for all $t \in T$, *i.e.*, the quasi-identifier attribute values for all individuals in the database. The adversary can easily learn these values from external databases and other resources.

**Definition 12** ($\delta$-disclosure privacy)**.** *We say that an equivalence class $\langle t \rangle$ is $\delta$-disclosure-private with regard to the sensitive attribute $S$ if, for all $s \in S$*

$$\mathcal{A}_{\mathsf{quot}}(\langle t \rangle) = \left|\log \frac{p(\langle t \rangle, s)}{p(T, s)}\right| < \delta$$

*A table $T$ is $\delta$-disclosure-private if for every $t \in \mathcal{E}_{\mathcal{Q}}$, $\langle t \rangle$ is $\delta$-disclosure private.*

Intuitively, a table is $\delta$-disclosure private if the distribution of sensitive attribute values within each quasi-identifier class is roughly the same as their distribution in the entire table. In contrast to [77], we use a multiplicative definition. It correctly models disclosures when some value of the sensitive attribute occurs in certain quasi-identifier classes, but not in others. It also allows us to derive a bound on the gain in adversarial knowledge, by relating the $\delta$ parameter to information gain used by decision-tree classifiers such as ID3 and C4.5. $\mathsf{Gain}(\mathcal{S}, \mathcal{Q})$ is defined as the difference between the entropy of $\mathcal{S}$ and the conditional entropy $H(\mathcal{S}|\mathcal{Q})$.

$$\mathsf{Gain}(\mathcal{S}, \mathcal{Q}) = H(\mathcal{S}) - H(\mathcal{S}|\mathcal{Q})$$

**Lemma 5.3.1.** *If $T$ satisfies $\delta$-disclosure privacy, then $Gain(\mathcal{S}, \mathcal{Q}) < \delta$. Let $\alpha_s = p(T, s)$ and let $\beta_{t,s} = p(\langle t \rangle, s)$. Note that $\alpha_s = \sum_{t \in \mathcal{E}_\mathcal{Q}} \frac{|\langle t \rangle|}{|T|} \beta_{t,s}$.*

Proof:

$$\begin{aligned}
& \mathrm{Gain}(\mathcal{S}, \mathcal{Q}) \\
= \ & \sum_{s \in S} -\alpha_s \log \alpha_s - \sum_{t \in \mathcal{E}_\mathcal{Q}} \frac{|\langle t \rangle|}{|T|} \sum_{s \in S} -\beta_{t,s} \log \beta_{t,s} \\
= \ & \sum_{s \in S} \sum_{t \in \mathcal{E}_\mathcal{Q}} -\frac{|\langle t \rangle|}{|T|} \beta_{t,s} \log \alpha_s - \sum_{t \in \mathcal{E}_\mathcal{Q}} \frac{|\langle t \rangle|}{|T|} \sum_{s \in S} -\beta_{t,s} \log \beta_{t,s} \\
= \ & \sum_{t \in \mathcal{E}_\mathcal{Q}} \frac{|\langle t \rangle|}{|T|} \sum_{s \in S} (-\beta_{t,s} \log \alpha_s + \beta_{t,s} \log \beta_{t,s}) \\
= \ & \sum_{t \in \mathcal{E}_\mathcal{Q}} \frac{|\langle t \rangle|}{|T|} \sum_{s \in S} \beta_{t,s} \log \frac{\beta_{t,s}}{\alpha_s} \\
< \ & \sum_{t \in \mathcal{E}_\mathcal{Q}} \frac{|\langle t \rangle|}{|T|} \sum_{s \in S} \beta_{t,s} \cdot \delta \quad = \quad \frac{\delta}{|T|} \sum_{t \in \mathcal{E}_\mathcal{Q}} |\langle t \rangle| \sum_{s \in S} \frac{|\langle t \rangle_s|}{|\langle t \rangle|} \\
= \ & \frac{\delta}{|T|} \sum_{t \in \mathcal{E}_\mathcal{Q}} \sum_{s \in S} |\langle t \rangle_s| \quad = \quad \delta
\end{aligned}$$

Lemma 5.3.1 shows that when a database satisfies $\delta$-disclosure privacy, the ability to build a predictor for sensitive attributes $\mathcal{S}$ based on the quasi-identifier $\mathcal{Q}$ is bounded by $\delta$. Note that definition 12 is *stronger* than the bound given by lemma 5.3.1, because it requires that the distributions $\mathcal{A}_{\mathsf{base}}$ and $\mathcal{A}_{\mathsf{san}}$ be similar, rather than just have similar entropies.

### 5.3.3 Measuring privacy of sanitized databases

Semantic privacy definitions, such as our Definition 12, bound sensitive attribute disclosure, but an actual database instance may have less sensitive attribute disclosure (and thus more privacy) than permitted by the definition.

Conventional privacy metrics rely on syntactic properties of the sanitized dataset: number of records with the same quasi-identifier ($k$-anonymity) or frequency of sensitive attributes within each quasi-identifier class ($\ell$-diversity). Unfortunately, the two metrics are incomparable. In [81], $k$ and $\ell$ are compared directly, even though the two have different domains: $k$ can vary from 1 to the total number of records, while $\ell$ can vary from 1 to the number of different sensitive attribute values. For example, a 1000-record database with a binary sensitive attribute can never be more than 2-diverse, but it can be anywhere up to 1000-anonymous.

We propose two different metrics to quantify attribute disclosure allowed by a sanitized database $T'$ as opposed to $T^*$ where all quasi-identifiers have been trivially suppressed. The first is based on the attribute disclosure distance $\mathcal{A}_{\mathsf{diff}}$:

$$\mathcal{A}_{\mathsf{know}} = \frac{1}{|T|} \sum_{t \in \mathcal{E}_Q} |\langle t \rangle| \cdot \mathcal{A}_{\mathsf{diff}}(\langle t \rangle)$$

$\mathcal{A}_{\mathsf{know}}$ stands for "adversarial knowledge gain." It is the average amount of information about the sensitive attributes of individual $t$ that the adversary learns because he is able to identify the class $\langle t \rangle$ based on $t$'s quasi-identifier.

One may also consider a metric based on $\mathcal{A}_{\mathsf{quot}}$, but only semantically private databases achieve a finite privacy score. Other privacy definitions allow sensitive attribute values to be absent from some quasi-identifier classes, enabling the adversary to learn with certainty that the corresponding individual does not have this value.

The second metric quantifies the adversary's ability to predict his target $t$'s sensitive attribute using his best strategy, which is to guess the most common sensitive attribute in $\langle t \rangle$. For a quasi-identifier class $\langle t \rangle$, let $s_{\mathsf{max}}(\langle t \rangle)$ be the most common sensitive attribute value found in $\langle t \rangle$. Then,

$$\mathcal{A}_{\mathsf{acc}} = \left( \frac{1}{|T|} \sum_{t \in \mathcal{E}_Q} |\langle t \rangle| \cdot p\left(\langle t \rangle, s_{\mathsf{max}}(\langle t \rangle)\right) \right) - p(T, s_{\mathsf{max}}(T))$$

$\mathcal{A}_{\mathsf{acc}}$ stands for "adversarial accuracy gain" and measures the increase in the adversary's accuracy after he observes the sanitized database $T'$ compared to his baseline accuracy from observing $T^*$, which is the most private database that can be obtained by generalization and suppression.

$\mathcal{A}_{\mathsf{acc}}$ *underestimates* the amount of information leaked by the sanitized table $T'$, because it does not consider shifts in the probabilities of non-majority sensitive attributes. It is still a useful metric because it can be directly compared to our metrics of data-mining utility, described in Section 5.4.

## 5.4 Measuring Utility

Utility of any dataset, whether sanitized or not, is innately tied to the computations that one may perform on it. For example, a census dataset may support an extremely accurate classification of income based on education, but not enable clustering based on household size. Without a workload context, it is meaningless to say whether a dataset is "useful" or "not useful," let alone to quantify its utility.

Nevertheless, the stated goal of privacy-preserving microdata publishing is to produce sanitized datasets that have "good" utility for a large variety of workloads. The unknown workload is an essential premise—if the workloads were known in advance, the data publisher could simply execute them on the original data and publish just the results instead of releasing a sanitized version of the data.

The need for a workload-independent measure of utility has led to the use of syntactic properties as a proxy for utility. One approach is to minimize the amount of generalization and suppression applied to the quasi-identifier attributes to achieve a given level of privacy [26]. This "minimization" is done with respect to absolute difference, relative distance, maximum distribution, or minimum suppression. Other syntactic metrics include the number of generalization steps, average size of quasi-identifier equivalence classes, the sum of squares of class sizes [81], and preservation of marginals [69].

Workload-independent metrics quantify the "damage" caused by sanitization, but they do not measure how much utility remains. For example, small quasi-identifier equivalence classes do not imply anything about the accuracy of classifiers that one may compute on the sanitized data [96].

It has been recognized that utility of sanitized databases must be measured empirically, in terms of specific workloads such as classification algorithms [63, 76, 120]. This does not necessarily contradict the "unknown workload" premise of sanitization. It simply acknowledges that even when sanitization satisfies a syntactic damage minimization requirement, it may still destroy the utility of a dataset for certain tasks; it is thus essential to measure the latter when evaluating effectiveness of various sanitization methods.

We can assume that users of the sanitized database are interested in workloads that take advantage of attribute correlations within the database, *e.g.*, construction of classifiers. For workloads which consider attributes in isolation, the data publisher can achieve maximum privacy by simply publishing two tables, one with the permuted quasi-identifiers, the other with the remaining attributes since they cannot be linked to the quasi-identifiers. Intuitively, utility of a sanitized database should be measured by how well cross-attribute correlations are preserved after sanitization.

It is critically important to measure *both* privacy and utility using the same methodology. Otherwise, maximizing utility may lead to privacy violations. For example, if utility is measured as the ability to predict sensitive attributes from the quasi-identifiers, then it is exactly the same as adversarial sensitive attribute disclosure! Iyengar [63] concludes that classification accuracy is maximized when attributes are *homogeneous* within each quasi-identifier group: this directly contradicts the *diversity* requirement [81, 114]. Similarly, [128] says that the data publishing process should preserve correlation between quasi-identifiers and sensitive attributes. This contradicts both diversity and semantic privacy, and immediately leads to sensitive attribute disclosure.

We aim to measure the tradeoffs between privacy and utility in a single framework, using semantic definitions for both: privacy in terms of adversarial sensitive attribute disclosure, utility in terms of concrete machine-learning tasks.

First, for a given workload $w$, we measure workload-specific utility of trivially sanitized datasets, *i.e.*, datasets from which either all quasi-identifiers $\mathcal{Q}$, or all sensitive attributes $\mathcal{S}$ have been removed. Both provide the maximum privacy achievable using generalization and suppression. Let $\mathcal{U}^{(w)}_{\mathsf{base}}$ be the corresponding empirical utility (to compute $\mathcal{U}^{(w)}_{\mathsf{base}}$, we pick the trivial sanitization with the largest utility). We give specific workloads and utility metrics in Section 5.5; for example, when the workload $w$ involves computing a classifier, $\mathcal{U}^{(w)}_{\mathsf{base}}$ is the accuracy of this classifier.

Then, we consider several non-trivially sanitized tables $T'$, one for each value of the sanitization parameter. For each table, we compute its workload-specific utility $\mathcal{U}_{\mathsf{san}}^{(w)}$.

The critical metric is $\mathcal{U}_{\mathsf{san}}^{(w)} - \mathcal{U}_{\mathsf{base}}^{(w)}$. This is the incremental utility gain provided by the release of non-trivially sanitized data. Note that if $\mathcal{U}_{\mathsf{san}}^{(w)} - \mathcal{U}_{\mathsf{base}}^{(w)}$ is close to 0, then non-trivial sanitization is pointless for this specific workload. In this case, a trivial sanitization which suppresses all $\mathcal{Q}$ or removes all $\mathcal{S}$ provides as much utility as any sophisticated sanitization algorithm while providing as much privacy as possible.

Another metric we'll employ is $\mathcal{U}_{\mathsf{max}}^{(w)}$, the utility of workload $w$ as measured on the original, pre-sanitization database. If $\mathcal{U}_{\mathsf{max}}^{(w)}$ is low (*e.g.*, the corresponding classifier has low accuracy), this means that the workload is inappropriate for the data regardless of sanitization. It does not make sense to measure utility in terms of this workload, because even if the users had been given the entire original database, the utility would have been low.

## 5.5 Experiments

Our experiments demonstrate that a trivial sanitizer which simply suppresses all quasi-identifiers or all sensitive attributes produces datasets with equivalent utility and better privacy (or equivalent privacy and better utility) than non-trivial generalization and suppression.

This appears to contradict previous work. For example, it was shown that useful machine-learning workloads can be evaluated on $k$-anonymous datasets [63, 76]. Of course, $k$-anonymity is neither necessary, nor sufficient for privacy. The "useful" datasets in question simply don't prevent sensitive attribute disclosure.

At the other end of the spectrum, $\ell$-diversity [81] and $t$-closeness [77] do limit sensitive attribute disclosure. Utility, however, is measured syntactically, by the number of generalization steps applied to quasi-identifiers, average size of quasi-identifier equivalence classes, sum of squares of class sizes, or preservation of marginals. In contrast to this chapter, the actual *data-mining* utility is not measured.

Wang *et al.* [120] give a sanitization which ensures a strong privacy definition *and* better data-mining utility on the UCI Adult dataset than simple removal of all sensitive attributes. They do not consider the other trivial sanitization, which is to remove all quasi-identifiers. We repeated their experiments and observed that their sanitization does not provide significantly better utility than the trivially sanitized dataset consisting of sensitive attributes only.

### 5.5.1 Achieving semantic privacy

Semantic privacy, as defined in Section 5.3.2, is easily incorporated into $k$-anonymity frameworks such as Incognito [74]. Like $\ell$-diversity [81] and $t$-closeness [77], semantic privacy has the *monotonicity property*: a generalization of a semantically private table is itself semantically private.

We used the implementation of generalization and suppression from

| Attribute | Values | Generalization |
|---|---|---|
| Age | 74 | continuous |
| Workclass | 7 | hierarchy |
| Education | 16 | continuous |
| Marital Status | 7 | hierarchy |
| Occupation | 14 | hierarchy |
| Race | 5 | hierarchy |
| Sex | 2 | hierarchy |
| Native Country | 41 | hierarchy |
| Salary | 2 | hierarchy |

Table 5.1: Summary of the UCI "Adult" dataset.

LeFevre *et al.* [76], and modified the constraint checking portion of the code to support recursive $(c, \ell)$-diversity ($c$=3 in all of our tests), $t$-closeness for nominal sensitive attributes, and semantic privacy from this chapter (in the figures, $s$ stands for $\delta$ from definition 12). This implementation is "workload-aware," *i.e.*, when choosing quasi-identifiers in $\mathcal{Q}$ to generalize, it attempts to maximize information gain for some target attribute.

### 5.5.2 Experimental methodology

To enable direct comparison with previous microdata sanitization work [77, 81], we used the *same* data for our experiments: the 45,222-record Adult database from the UCI Machine Learning Repository [7], described in Table 5.1. Our classifier learning used Weka with the default settings for C4.5 (J48), Random Forests, and Naive Bayes. For all classification experiments, we used 10-fold cross-validation.

**Choosing the quasi-identifier.** In a real database, the set of quasi-identifier attributes $\mathcal{Q}$ is domain-specific, and includes the attributes to which the adversary is most likely to have access via an external database (*e.g.*, demographic information). For our experiments, we examined several different sets of attributes for $\mathcal{Q}$. All were picked to maximize the likelihood that sanitization will produce a useful table.

It is common in the literature to choose large quasi-identifiers, sometimes consisting of all non-sensitive attributes. A larger quasi-identifier, however, gives more prior information to the adversary and requires heavier generalization and suppression during sanitization. Large quasi-identifiers thus underestimate utility of the dataset and increase the risk of a privacy breach. Our most important criterion for choosing $\mathcal{Q}$ was to keep it *small*, to make the adversary's task as hard as possible.

Furthermore, if a legitimate user (whom we will call "researcher") is to get more utility out of the sanitized database than the adversary, his task(s) must be different from the adversary's. If the sensitive attribute is also the researcher's target attribute and all other attributes are quasi-identifiers, then both the researcher and the attacker are trying to use $\mathcal{Q}$ to predict $\mathcal{S}$! This is why in our measurements of utility, we consider utility of classification on "neutral" attributes which are neither quasi-identifiers, nor sensitive.

**Choosing the workload and the sensitive attribute.** We must also choose a workload for the legitimate researcher. As discussed in Section 5.4, classification is a good workload because quality of classification depends on the correlations between attributes in the database, and the entire purpose of "truthfully" publishing quasi-identifiers and sensitive attributes *together* is to preserve these cross-attribute correlations.

We will look at classification of both sensitive and neutral attributes. It is important to choose a workload (target) attribute $v$ for which the presence of the quasi-identifier attributes $\mathcal{Q}$ in the sanitized table actually matters. If $v$ can be learned equally well with or without $\mathcal{Q}$, then the data publisher can simply suppress all quasi-identifiers.

Table 5.2 shows the difference in decision-tree learning accuracy depending on whether or not the quasi-identifier (age, sex, race) is included. Only marital status shows a significant drop when the quasi-identifier is entirely suppressed, thus we choose it as the workload attribute for the "Occupation" dataset. Even though salary is intuitively the sensitive attribute in the "Adult" dataset, when the workload $w$ is "learning salary," then $\mathcal{U}^{(w)}_{\mathsf{max}} \approx \mathcal{U}^{(w)}_{\mathsf{base}}$. Since we are interested in measuring utility of non-trivial sanitization (*i.e.*, how much utility it provides over the table in which all quasi-identifiers have been suppressed), we are only interested in scenarios where $\mathcal{U}^{(w)}_{\mathsf{san}} > \mathcal{U}^{(w)}_{\mathsf{base}}$; otherwise, complete suppression of $\mathcal{Q}$ provides better privacy and same utility as any non-trivial sanitization. When $\mathcal{U}^{(w)}_{\mathsf{max}} \approx \mathcal{U}^{(w)}_{\mathsf{base}}$, it cannot be the case that $\mathcal{U}^{(w)}_{\mathsf{san}} > \mathcal{U}^{(w)}_{\mathsf{base}}$, thus we do *not* choose salary as the sensitive attribute, and use marital status instead.

**Datasets used.** In the "Marital" dataset, $\mathcal{Q}$=(age, occupation, education),

$\mathcal{S}$=marital status, the workload attribute is salary. In the "Occupation" dataset, $\mathcal{Q}$=(age, sex, race), $\mathcal{S}$=occupation, the workload attribute is marital status.

### 5.5.3 Experimental results

**Learning the sensitive attribute $\mathcal{S}$.** The researcher may wish to build a classifier for the sensitive attribute $\mathcal{S}$ using both the quasi-identifiers and the neutral attributes as predictors. Of course, if sanitization has been correctly performed, it is impossible to build a good classifier for $\mathcal{S}$ based *only* on $\mathcal{Q}$, because good sanitization must destroy any correlation between $\mathcal{S}$ and $\mathcal{Q}$ (otherwise, the adversary will easily learn the sensitive attributes associated with any quasi-identifier). Our results demonstrate that the researcher can build a classifier for $\mathcal{S}$ without using the attributes in $\mathcal{Q}$ just as well as when using sanitized versions of $\mathcal{Q}$.

Figure 5.1 shows the loss of privacy, measured as the gain in the accuracy of adversarial classification $\mathcal{A}_{\mathsf{acc}}$ for different sanitizations of the "Marital"

| Attribute | Intact | Suppressed |
|---|---|---|
| Workclass | 74.8618% | 74.6672% |
| Education | 41.6899% | 41.1658% |
| Marital Status | 69.3623% | 58.5777% |
| Occupation | 32.2387% | 30.0363% |
| Country | 91.7960% | 91.6147% |
| Salary | 82.7916% | 82.4311% |

Table 5.2: The effect of including age, sex, and race on decision-tree learning accuracy.

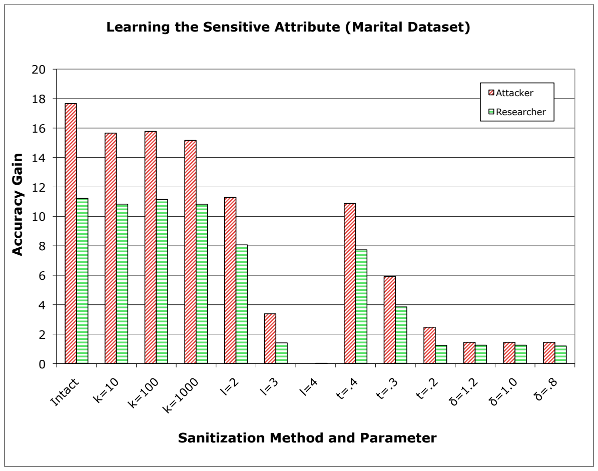


Figure 5.1: Gain in classification accuracy for the sensitive attribute (marital) in the "Marital" dataset. With trivial sanitization, accuracy is 46.56% for the adversary and 58.30% for the researcher.

dataset, and compares it with the gain in workload utility $\mathcal{U}^{(w)}_{\mathsf{san}} - \mathcal{U}^{(w)}_{\mathsf{base}}$ where the workload $w$ is building a decision-tree classifier for the "marital status" attribute. As explained in Section 5.3.3, accuracy of adversarial classification *underestimates* the actual amount of sensitive attribute disclosure. Figure 5.1 shows that releasing a sanitized table instead of simply suppressing all $\mathcal{Q}$ helps the adversary associate sensitive attributes with individuals much more than it helps the researcher to build legitimate classifiers. Figure 5.2 shows the same result for the "Occupation" dataset, where the workload $w$ is building a decision-tree classifier for the "occupation" attribute.

**Learning a non-sensitive workload attribute.** Perhaps it is not surprising that sanitization makes it difficult to build an accurate classifier for the

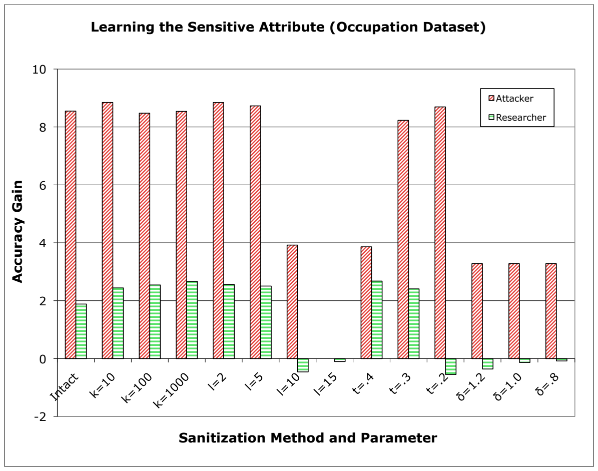


Figure 5.2: Gain in classification accuracy for the sensitive attribute (occupation) in the “Occupation” dataset. With trivial sanitization, accuracy is 13.31% for the adversary and 30.18% for the researcher.

sensitive attribute. We now consider the case when the researcher wishes to build a classifier for a *non-sensitive* attribute $v$.

If both $\mathcal{Q}$ and $\mathcal{S}$ are correlated with $v$, then a classifier based on both $\mathcal{Q}$ and $\mathcal{S}$ may have higher accuracy than one that considers only $\mathcal{Q}$, only $\mathcal{S}$, or neither. Our results show that sanitization which removes the correlation between $\mathcal{S}$ and $\mathcal{Q}$ also destroys the correlation between $\mathcal{S}$ and $v$.

In these experiments, we compute $\mathcal{U}^{(w)}_{\mathsf{base}}$ by running different machine learning algorithms on both trivially sanitized versions of the database; $\mathcal{U}^{(w)}_{\mathsf{base}}$ is the accuracy of the best classifier. We then compute $\mathcal{U}^{(w)}_{\mathsf{base}} - \mathcal{U}^{(w)}_{\mathsf{san}}$ for different sanitizations and different machine learning algorithms, and compare this to the increase in adversarial accuracy $\mathcal{A}_{\mathsf{acc}}$.

Figure 5.3 compares gains in adversary's and researcher's respective accuracies for the "Marital" dataset (workload $w$ is learning the "salary" attribute). The classification accuracies with the sensitive attribute removed were 80.73% for J48, 77.12% for Random Forests, and 79.45% for Naive Bayes. Thus, the baseline for utility was set to 80.73%.

Figure 5.4 compares gains in adversary's and researcher's respective accuracies for the "Occupation" dataset (workload $w$ is learning the "marital status" attribute). Here we see that $\mathcal{U}^{(w)}_{\mathsf{max}} = 69.52\%$ and $\mathcal{U}^{(w)}_{\mathsf{base}} = 69.30\%$ where the baseline comes from J48 learning with the sensitive attribute removed. With such a small gap between $\mathcal{U}^{(w)}_{\mathsf{max}}$ and $\mathcal{U}^{(w)}_{\mathsf{base}}$, it is not surprising that classification accuracies for sanitized datasets are below those of trivial sanitizations, where the sensitive attribute was simply removed.

**Privacy of the sanitized database.** Table 5.3 shows the $\mathcal{A}_{\mathsf{acc}}$ and $\mathcal{A}_{\mathsf{know}}$ scores for different sanitizations of the "Occupation" dataset (the accuracies are low even for the intact database because the quasi-identifiers do not identify a unique individual). Even for large $k$, $k$-anonymity barely changes the value of $\mathcal{A}_{\mathsf{know}}$ compared to the intact database. In other words, **$k$-anonymity provides no privacy improvement whatsoever** on this dataset. Furthermore, **$\ell$-diversity is no better than trivial sanitization** because it requires complete suppression of the quasi-identifiers to substantially limit the gain in adversary's knowledge.

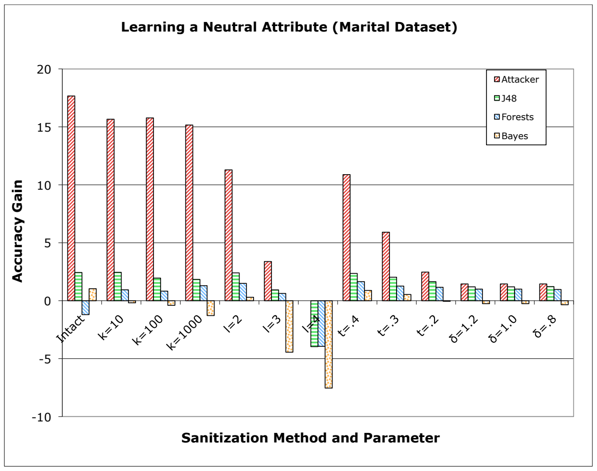


Figure 5.3: Gain in the adversary's ability to learn the sensitive attribute (marital) and the researcher's ability to learn the workload attribute (salary) for the "Marital" dataset. With the trivial sanitization, accuracy is 46.56% for the adversary, and 80.73% for the researcher.

## 5.6 Achieving Privacy and Utility

Our experimental results in Section 5.5 indicate that, empirically, it is difficult to find a database table on which sanitization permits both privacy and utility. Any incremental utility gained by non-trivial sanitization (as opposed to simply removing quasi-identifiers or sensitive attributes) is more than offset by a decrease in privacy, measured as the adversarial sensitive attribute disclosure. It is possible, however, to construct an *artificial* database, for which sanitization provides both complete utility and complete privacy, even for the strongest definition of privacy (semantic privacy).

Consider table $T$, in which each tuple $t$ has five attributes $a_1, a_2, a_3, a_4, a_5$.

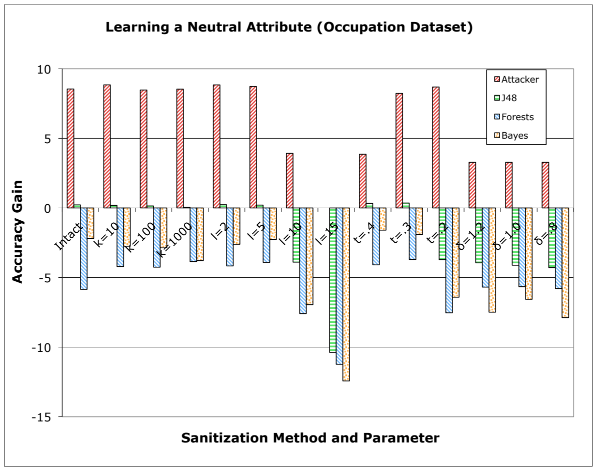


Figure 5.4: Gain in the adversary's ability to learn the sensitive attribute (occupation) and the researcher's ability to learn the workload attribute (marital) for the "Occupation" dataset. With the trivial sanitization, accuracy is 13.31% for the adversary, and 69.30% for the researcher.

Their values are defined by three coin flips $r_1, r_2, r_3$, which are generated independently at random for each tuple. The attributes are as follows:

$$a_1 = r_1 \quad a_2 = r_2 \quad a_3 = (r_2, r_3) \quad a_4 = r_1 \oplus r_3 \quad a_5 = r_1 \oplus r_2$$

Now consider the case where $\mathcal{Q} = \{a_1, a_2\}$, $\mathcal{S} = a_3$. This database is a candidate for sanitization, since $\mathcal{Q}$ provides a lot of information about $\mathcal{S}$ (half of the sensitive attribute can be predicted perfectly from the quasi-identifier). If we sanitize by suppressing $a_2$, then we are left with a database $T'$ which is perfectly private, since $a_1$ reveals nothing about $a_3$. But this database also has perfect utility, since a researcher can learn $a_4$ exactly from $a_1$ and $a_3$, and he can learn $a_5$ exactly from $a_1$ and $a_3$, and he can learn $a_3$ exactly from $a_1, a_4,$

| Sanitization | $\mathcal{A}_{\mathsf{acc}}$ | $\mathcal{A}_{\mathsf{know}}$ |
|---|---|---|
| Intact | 0.1034 | 0.2492 |
| k=10 | 0.0957 | 0.2331 |
| k=100 | 0.0909 | 0.2236 |
| k=1000 | 0.0885 | 0.2131 |
| l=2 | 0.0966 | 0.2353 |
| l=5 | 0.0940 | 0.2316 |
| l=10 | 0.0400 | 0.1217 |
| l=15 | 0 | 0 |
| t=.4 | 0.0924 | 0.2264 |
| t=.3 | 0.0861 | 0.2131 |
| t=.2 | 0.0396 | 0.1213 |
| $\delta$=1.2 | 0.0328 | 0.0944 |
| $\delta$=1.0 | 0.0327 | 0.0937 |
| $\delta$=.8 | 0.0327 | 0.0915 |
| Suppressed | 0 | 0 |

Table 5.3: $\mathcal{A}_{\mathsf{acc}}$ and $\mathcal{A}_{\mathsf{know}}$ scores for different sanitizations of the "Occupation" dataset.

and $a_5$. Furthermore, if $\mathcal{Q}$ were completely suppressed, the researcher could learn nothing about $a_4$ or $a_5$ and he could only learn half the information about $a_3$ $(r_2 \oplus r_3)$. If $\mathcal{S}$ were omitted, the researcher could learn nothing about $a_4$ and only half of the information about $a_5$.

This artificial dataset is very unusual, and it is unclear whether any real datasets exhibit similar properties. For instance, sensitive attributes $\mathcal{S}$ can be split into two parts, one of which is 100% correlated with the quasi-identifiers $\mathcal{Q}$ and the other is completely independent of $\mathcal{Q}$. Sanitization can thus suppress the dependent part of $\mathcal{Q}$ entirely, while leaving the independent part intact. Furthermore, $a_4$ and $a_5$ are both completely determined by the joint distribution of $\mathcal{S}$ and $\mathcal{Q}$, but independent of either one taken alone. It is unclear how often attributes which are pairwise independent but jointly dependent arise in real data.

## 5.7 Conclusions

Microdata privacy can be understood as prevention of membership disclosure (the adversary should not learn whether a particular individual is included in the database) or sensitive attribute disclosure (the sanitized database should not reveal very much information about any individual's sensitive attributes). It is known that generalization and suppression cannot prevent membership disclosure [38, 95]. For sensitive attribute disclosure, perfect privacy can be achieved—against a very weak adversary who knows just the quasi-identifiers—by simply removing the sensitive attributes or the quasi-identifiers from the published data. Of course, these trivial sanitizations also destroy any utility that depended on the removed attributes.

Algorithms such as $k$-anonymity and $\ell$-diversity leave all sensitive attributes intact and apply generalization and suppression to the quasi-identifiers. The goal is to keep the data "truthful" and thus provide good utility for data-mining applications, while achieving less than perfect privacy. We argue that utility is best measured by the success of data-mining algorithms such as decision-tree learning which take advantage of relationships between attributes. Algorithms that need only aggregate statistical information can be executed on perturbed or randomized data, with much stronger privacy guarantees against stronger adversaries than achieved by $k$-anonymity,

$\ell$-diversity, and so on.

Our experiments, carried out on the same UCI data as was used to validate existing microdata sanitization algorithms, show that the privacy vs. utility tradeoff for these algorithms is very poor. Depending on the sanitization parameter, sanitized datasets either provide no additional utility vs. trivial sanitization, or the adversary's ability to compute the sensitive attributes of any individual increases much more than the accuracy of legitimate machine-learning workloads.

An important question for future research is whether there exists any real-world dataset on which quasi-identifier generalization supports meaningfully better data-mining accuracy than trivial sanitization without severely compromising privacy via sensitive attribute disclosure.

Another important question is how to design microdata sanitization algorithms that provide both privacy and utility. Sensitive attribute disclosure results, in part, from the fact that each individual $t$ can only belong to a unique quasi-identifier equivalence class $\langle t \rangle$ in the sanitized table $T'$. This is a consequence of the requirement that the generalization hierarchy be totally ordered [26]. This requirement helps the adversary, but does not improve utility. If we consider $G(t)$, the set of records in $T'$ whose quasi-identifier values are generalizations of $t[\mathcal{Q}]$, there is no privacy reason why each record of $G(t)$ must have the same quasi-identifier values. It is possible that a generalization strategy that uses, *e.g.*, DAGs instead of totally ordered hierarchies may provide better privacy than the existing algorithms.

# Chapter 6

# Privacy-Preserving Graph Algorithms

## 6.1 Introduction

In this chapter, we investigate scenarios with two mutually distrustful parties, each in possession of a graph (representing, *e.g.*, a network topology, a distribution channel map, or a social network). The parties wish to compute some algorithm on their *combined* graph, but do not wish to reveal anything about their private graphs beyond that which will be necessarily revealed by the output of the algorithm in question.

For example, consider two Internet providers who are contemplating a merger and wish to see how efficient the resulting joint network would be without revealing the details of their existing networks; or two transportation companies trying to determine who has the greatest capacity to ship goods between a given pair of cities without revealing what that capacity is or which distribution channels contribute to it; or two social networking websites wishing to calculate aggregate statistics such as degrees of separation and average number of acquaintances without compromising privacy of their users, and so on.

We construct privacy-preserving versions of classic graph algorithms for APSD (all pairs shortest distance) and SSSD (single source shortest distance). Our algorithm for APSD is new, while the SSSD algorithm is a privacy-preserving transformation of the standard Dijkstra's algorithm. We also show that minimum spanning trees can be easily computed in a privacy-preserving manner. As one of our tools, we develop protocols for privacy-preserving set union, which are results of independent interest.

We demonstrate that our constructions are significantly more efficient than those based on generic constructions for secure multi-party computation such as Yao's garbled circuits [132]. Some of the efficiency gain is due to our use of canonical orderings on graph edges. We believe that this technique may find applicability beyond the problems considered in this chapter.

We prove that our constructions are secure in the *semi-honest* model. Assuming that a party correctly follows the protocol, there is no efficient adversary that can extract more information from the transcript of the protocol execution than is revealed by that party's private input and the result of the graph algorithm. Our choice of the semi-honest model follows previous work on privacy-preserving data mining such as Lindell and Pinkas' construction for a privacy-preserving version of the ID3 decision tree learning algorithm [78], and constructions by Yang *et al.* for privacy-preserving classification [130].

In general, the semi-honest model seems to be the right fit for our setting, where there is no realistic way to verify that the parties are submitting their true graphs as private inputs. The best we could hope for in the case of actively malicious participants is a protocol in which the parties first commit to their graphs, and then prove at every step of the protocol that their inputs match their commitments. This would greatly complicate the protocols without providing any protection against parties who maliciously choose their graphs in such a way that the result of the computation on the joint graph completely reveals the other party's input. We leave investigation of privacy-preserving graph algorithms in the model with malicious participants to future work.

## 6.2 Tools

As building blocks for our algorithms, we use protocols for privacy-preserving computation of a minimum $\min(x, y)$ and set union $S_1 \cup S_2$.

In the minimum problem, the parties have as their respective private inputs integers $x_1$ and $x_2$ which are representable in $n$ bits. They wish to privately compute $m = \min(x_1, x_2)$. Because this problem is efficiently solved by a simple circuit containing $O(n)$ gates, it is a good candidate for Yao's generic method [132]. An implementation of this functionality with Yao's garbled circuit requires 2 communication rounds with $O(n)$ total communication complexity and $O(n)$ computational complexity.

### 6.2.1 Privacy-preserving set union

In the set union problem, parties $P_1$ and $P_2$ have as their respective private inputs sets $S_1$ and $S_2$ drawn from some finite universe $U$. They wish to compute the set $S = S_1 \cup S_2$ in a privacy-preserving manner, *i.e.*, without leaking which elements of $S$ are in the intersection $S_1 \cap S_2$. We will define $|S_1| = s_1$, $|S_2| = s_2$, $|S| = s$, and $|U| = u$.

In this section, we give two of our own solutions for privacy-preserving set union: the iterative method, and the tree-pruning method. Both require communication and computational complexity that is logarithmic in $u$, provided $s$ is small (note that even if we are not concerned about privacy, computing the set union requires at least $O(s \lg u)$ bandwidth, although it can be done in 1 round). We also survey several previously proposed techniques that can be used to compute the set union, but these techniques are all either linear in $u$ (or worse), or do not fully preserve privacy.

**Iterative method.** The basic idea of the iterative method is to build up $S$ one element at a time, from "smallest" to "largest." Before the protocol begins, both parties agree upon a canonical total ordering for the entire universe $U$. As a result, each element in $U$ is given an integer label with $\lg u$ bits. In addition, we need a label representing $\infty$, for which can simply use the integer $u + 1$. The protocol proceeds as follows:

*Step 1.* Set $S = \emptyset$.

*Step 2.* $P_1$ selects $m_1$ as the canonically smallest element in $S_1$, or sets $m_1 = \infty$ if $S_1 = \emptyset$. $P_2$ likewise selects $m_2$ as the canonically smallest element in $S_2$, or sets $m_1 = \infty$ if $S_1 = \emptyset$.

*Step 3.* Using a protocol for private minimum, $P_1$ and $P_2$ privately compute $m = \min(m_1, m_2)$.

*Step 4.* If $m = \infty$, stop and return $S$. Otherwise, $S = S \cup \{m\}$ and the parties remove $m$ from their input sets (it may be present in one or both). Then return to step 2.

The protocol preserves privacy because, given the output set $S$, a simulator can determine the value of $m$ at each iteration. The protocol used for computing the minimum is private, so there exists an efficient algorithm that can simulate its execution to the party $P_1$ given its input and the output $m$ (likewise for $P_2$). The simulator for the iterative method protocol uses the simulator for the minimum protocol as a subroutine, following the standard hybrid argument.

The iterative method protocol requires $s + 1$ iterations, and in each iteration the minimum of two $(\lg u)$-bit integers is privately computed. Using Yao's method, this requires a circuit with $2 \lg u$ inputs and $O(\lg u)$ gates. The $2 \lg u$ oblivious transfers can all take place in parallel, and since Yao's method requires a constant number of rounds the whole protocol takes $O(s)$ communication rounds. The total communication and computational complexity for the iterative method is $O(s \lg u)$.

**Tree-pruning method.** Before the tree-pruning protocol begins, the participants agree on a $(\lg u)$-bit binary label for each element in the universe (note that a canonical total ordering would automatically provide such a label). The basic idea of the protocol is that the participants will consider label prefixes of increasing length, and use a privacy-preserving Bit-Or protocol (see Section 6.2.2) to determine if either participant has an element with that prefix in his set.

Initially, the single-bit prefixes “0” and “1” are set “live.” The protocol proceeds through $\lg u$ rounds, starting with round 1. In the $i$th round, the participants consider the set P of $i$-bit “live” prefixes. For each prefix $p \in P$, each participant sets his respective 1-bit input to 1 if he has an element in his set with prefix $p$, and to 0 if he does not have any such elements. The participants then execute a privacy-preserving Bit-Or protocol on their respective 1-bit inputs. If the result of the Bit-Or protocol is 1, then $p0$ and $p1$ are set as live $(i+1)$-bit prefixes. Otherwise, $p0$ and $p1$ are dead prefixes.

By a simple inductive argument, the number of live prefixes in each round does not exceed $2 \cdot |S|$, because an $i$-bit prefix $p_i = b_1 \ldots b_i$ can be live if and only if at least one of the participants has an element whose label starts with $b_1 \ldots b_{i-1}$, and the number of such elements cannot exceed the total number of elements in the union, *i.e.*, $|S|$.

In the last round ($i = \lg u$), the length of the prefix is the same as the length of the binary labels, and the entire set $P$ of live prefixes is declared to be the output $S$ of the privacy-preserving set union protocol.

The tree-pruning protocol preserves privacy because, given the output set $S$, a simulator can determine the output of each of the Bit-Or protocols. As in the case of the iterative method protocol, we can construct a simulator for the tree-pruning protocol that uses a simulator for the Bit-Or protocol as a subroutine, and prove its correctness using a hybrid argument. The construction is simple and is omitted for brevity.

The tree-pruning protocol requires $\lg u$ iterations, and in each iteration the pairwise BIT-OR of at most $2s$ bits is computed. These computations can all take place in parallel, so the protocol requires $O(\lg u)$ communication rounds. Each iteration requires $O(s)$ communication and computational complexity, so the entire protocol has complexity $O(s \lg u)$. Both the iterative method and tree pruning protocols have the same complexity, but different numbers of rounds. The iterative method requires fewer rounds when $s = o(\lg u)$.

### Survey of privacy-preserving set union protocols

**Generic Yao's method.** It is easy to construct a circuit for computing the set union. Each party $P_p$ inputs one bit for every element $e$ in the universe $U$. The input bit $b_{pi}$ is set to 1 if party $P_p$ has element $e_i$ in his set, and 0 otherwise. The circuit consists of $|U|$ AND gates, each of which takes as inputs $b_{0i}$ and $b_{1i}$ and outputs $o_i = b_{0i} \wedge b_{1i}$. Then $o_i = 1$ iff element $e_i$ is in the set union. Since this circuit has $O(u)$ inputs and $O(u)$ gates, we conclude that the computational overhead and the communication complexity are both $O(u)$.

**Commutative encryption.** Clifton *et al.* [27] present a simple construction for privacy-preserving set union that uses commutative encryption. Each party encrypts the elements in its set, exchanges the encrypted sets with the other party, and then encrypts the other party's encrypted elements with its own key. The double-encrypted sets are then combined. Due to commutativity of encryption, all elements in the intersection appear as duplicates. They are removed, and the remaining elements are decrypted. Scrambling the order of elements may hide which elements are in the intersection, but the size of the intersection is still revealed, thus this method is not secure in the standard sense of definition 3. This protocol requires communication and computational complexity $O(|s_1| + |s_2|)$.

**Complement of set intersection.** When the universe $U$ is small, it is possible to use complementation and take advantage of the fact that $S_1 \cup S_2 = \overline{\bar{S_1} \cap \bar{S_2}}$. Freedman *et al.* [48] present a privacy-preserving protocol for set intersection that uses homomorphic encryption which requires $O(k)$ communication overhead and $O(k \ln \ln k)$ computation overhead, where $k$ is the size of the set intersection. For applications considered in this chapter, sets $S_1$ and $S_2$ are very small, so their complements are of size $O(u)$. As a result, this method requires $O(u \ln \ln u)$ computation, which is unacceptable.

**Polynomial set representation.** Kissner and Song [71] present a method for representing sets as polynomials, and give several privacy-preserving protocols for set operations using these representations. They do not provide a protocol for the standard set union problem. Instead, they give a protocol for the "threshold set union" problem, in which the inputs are multi-sets and the output is the set of elements whose multiplicity of appearance in the union exceed some threshold; the intersection of the input sets is also revealed. When applied to regular sets (as opposed to multi-sets) this protocol does not preserve privacy as the intersection is the only information one can hope to keep private.

### 6.2.2 Privacy-preserving bit-or

First, observe that the circuit for computing OR of 2 bits consists in a single gate. Therefore, even the generic construction using Yao's protocol [132] is efficient, requiring a single *1-out-of-2* oblivious transfer.

An alternative construction without oblivious transfers is provided by a semantically secure homomorphic encryption scheme such as ElGamal. Suppose Alice and Bob want to compute OR of their respective bits $b_A$ and $b_B$ in a privacy-preserving manner (Alice and Bob are honest, but curious). Alice picks some cyclic group $G$ of prime order $q$ with generator $g$ where the Decisional Diffie-Hellman problem is presumed hard, *e.g.*, the group of quadratic residues modulo some large prime $p = 2q + 1$, and chooses its secret key $k$ at random from $\{0, \ldots, q-1\}$. Alice sends to Bob its public key $q, g, g^k$ together with its ciphertext $c_A$, which is created as follows. If $b_A = 0$, then $c_A = (g^r, g^{kr})$, where $r$ is randomly selected from $\{0, \ldots, q-1\}$. If $b_A = 1$, then $c_A = (g^r, g \cdot g^{kr})$.

Upon receipt of $c_A = (\alpha, \beta)$ and Alice's public key, Bob computes $c_B$ as follows. First, it randomly picks $r' \in \{0, \ldots, q-1\}$. If $b_B = 0$, then

$c_B = (\alpha^{r'}, \beta^{r'})$. If $b_B = 1$, then $c_B = (\alpha^{r'}, g^{r'} \cdot \beta^{r'})$. Bob returns $c_B$ to Alice.

Alice computes bit $b$ by decrypting $c_B = (\gamma, \delta)$ with its private key $k$, *i.e.*, $b = \frac{\delta}{\gamma^k}$. Clearly, if $b_A = b_B = 0$, then $b = 1$. In this case, Alice declares that $b_A \vee b_B = 0$. If $b \neq 1$, then Alice declares that $b_A \vee b_B = 1$.

To verify that this construction preserves privacy, observe that secrecy of $b_A$ follows from the semantic security of ElGamal. Now suppose $b_A = 1$. If $b_B = 0$, then the decrypted plaintext $b = g^{r'}$. If $b_B = 1$, then $b = g^{2r'}$. Since $B$ does not know $r'$, it cannot tell the difference. Thus, $A$ does not learn $b_B$ if $b_A = 1$.

## 6.3 Privacy-Preserving Algorithms on Joint Graphs

We now present our constructions that enable two parties to compute algorithms on their joint graph in a privacy-preserving manner. Let $G_1$ and $G_2$ be the two parties' respective weighted graphs. Assume that $G_1 = (V_1, E_1, w_1)$ and $G_2 = (V_2, E_2, w_2)$ are complete graphs on the same set of vertices, that is, $V_1 = V_2$ and $E_1 = E_2$. Let $w_1(e)$ and $w_2(e)$ represent the *weight* of edge $e$ in $G_1$ and $G_2$, respectively. To allow incomplete graphs, the excluded edges may be assigned weight $\infty$. We are interested in computing algorithms on the parties' joint *minimum* graph $\text{gmin}(G_1, G_2) = (V, E, w_{min})$ where $w_{min}(e) = \min(w_1(e), w_2(e))$, since minimum joint graphs seem natural for application scenarios such as those considered in section 6.1.

### 6.3.1 All pairs shortest distance

The All Pairs Shortest Distance (APSD) problem is the classic graph theory problem of finding shortest path distances between all pairs of vertices in a graph (see, *e.g.*, [29]). We will think of APSD($G$) as returning a complete graph $G' = (V, E', w')$ in which $w'(e_{ij}) = d_G(i, j)$ and $V$ is the original edge set of $G$. Here $d_G(i, j)$ represents the shortest path distance from $i$ to $j$ in $G$. This problem is particularly well suited to privacy-preserving computation because the solution "leaks" useful information that can be used by the simulator.

To motivate the problem, consider two shipping companies who are hoping to improve operations by merging so that they can both take advantage of fast shipping routes offered by the other company. They want to see how quickly the merged company would be able to ship goods between pairs of cities, but they don't want to reveal all of their shipping times (and, in particular, their inefficiencies) in case the merger doesn't happen. In other words, they wish to compute APSD($G$) where $G = \mathrm{gmin}(G_1, G_2)$.

The basic idea behind our construction is to build up the solution graph by adding edges in order from shortest to longest. The following algorithm takes as input the parties' complete graphs $G_1$ and $G_2$. The graphs may be directed or undirected, but they must have strictly positive weight functions.

1. For notational convenience we introduce a variable $k$, initially set to 1, that represents the iteration count of the algorithm. Color each edge in $E$ "blue" by letting $B^{(k)}$ denote the set of blue edges in the edge set $E$

at iteration $k$, and setting $B^{(0)} = E$. Let $R^{(k)}$ denote the set of "red" edges, $R^{(k)} \overset{def}{=} E - B^{(k)}$. The lengths of red edges have reached their final values and will not change as the algorithm proceeds, while the lengths of blue edges may still decrease.

2. A public graph $G_0^{(0)} = (V, E, w_0^{(0)})$ is created. Its edges are all initially weighted as $w_0^{(0)}(e) = \infty$. When the algorithm terminates after $n$ iterations, we will have $w_0^{(n)}(e_{ij}) = d_G(i, j)$ and $B^{(n)} = \emptyset$.

3. The parties compute the following public value

$$m_0^{(k)} = \min_{e \in B^{(k-1)}} w_0^{(k-1)}(e) \tag{6.1}$$

   and the respective private values

$$\begin{aligned} m_1^{(k)} &= \min_{e \in B^{(k-1)}} w_1(e), \text{ and} \qquad (6.2)\\ m_2^{(k)} &= \min_{e \in B^{(k-1)}} w_2(e) \qquad (6.3) \end{aligned}$$

4. Now the parties *privately* compute the length of the smallest blue edge among all three graphs, $m^{(k)} = \min(\min(m_1^{(k)}, m_0^{(k)}), \min(m_2^{(k)}, m_0^{(k)}))$, using a generic protocol for private minimum (section 6.2). This protocol does not reveal the larger value.

5. The parties form the following public set

$$S_0^{(k)} = \{e | w_0^{(k-1)}(e) = m^{(k)}\} \tag{6.4}$$

and the respective private sets

$$S_1^{(k)} = \{e|w_1(e) = m^{(k)}\}, \text{ and} \tag{6.5}$$

$$S_2^{(k)} = \{e|w_2(e) = m^{(k)}\} \tag{6.6}$$

By construction, $S_0^{(k)}$, $S_1^{(k)}$, and $S_2^{(k)}$ contain only blue edges.

6. First, the parties *privately* compute the set union $S^{(k)} = S_0^{(k)} \cup S_1^{(k)} \cup S_2^{(k)}$. This is done using the privacy-preserving set union algorithm from section 6.2. Next, the color of each edge $e \in S^{(k)}$ is changed from blue to red by setting $B^{(k)} = B^{(k-1)} - S^{(k)}$. Define a weight function $w_0'^{(k)}$ by
$$w_0'^{(k)}(e) = \begin{cases} m^{(k)} & \text{if } e \in S^{(k)} \\ w_0^{(k-1)}(e) & \text{otherwise} \end{cases} \tag{6.7}$$

7. Examine triangles with an edge $e_{ij} \in S^{(k)}$, an edge $e_{jk} \in R^{(k)}$, and an edge $e_{ik} \in B^{(k)}$. Define the weight function $w_0^{(k)}$ by fixing these triangles if they violate the triangle inequality under $w_0'^{(k)}$. More precisely, if $w_0'^{(k)}(e_{ij}) + w_0'^{(k)}(e_{jk}) < w_0'^{(k)}(e_{ik})$, then define $w_0^{(k)}(e_{ik}) = w_0'^{(k)}(e_{ij}) + w_0'^{(k)}(e_{jk})$. Do the same for triangles with an edge $e_{ij} \in R^{(k)}$, an edge $e_{jk} \in S^{(k)}$, and an edge $e_{ik} \in B^{(k)}$.

8. If there are still blue edges, go to step 3. Otherwise stop; the graph $G_0^{(k)}$ holds the solution to APSD($G$).

The algorithm is proved correct in Section 6.5. The proof of privacy follows.

*Privacy.* We describe a simulator for $P_1$; the simulator is given $P_1$'s input to the protocol, $x$, and the output of the protocol, $f(x, y) = G'$. The simulators are identical for $P_1$ and $P_2$ except for the asymmetry in the simulation of the set union and minimum subprotocols. We assume that simulators for the subprotocols exist because they are private protocols. For instance, if Yao's protocol is used then we can use the simulator in [80].

We will assume that there are $n$ protocol rounds. The view of $P_1$ is

$$\{RT^m(x_1, y_1), RT^u(x_2, y_2), RT^m(x_3, y_3), \ldots, RT^u(x_{2n}, y_{2n})\} \tag{6.8}$$

where $RT^m$ denotes the *real transcript* of the private minimum protocol, and $RT^u$ denotes the real transcript of the private set union protocol.

We will show in later theorems that the output of each of these protocol executions can be computed by the simulator as a polynomial function of $G'$, which we will denote as $h_i^m(G')$ and $h_i^u(G')$. We will also show that $P_1$'s input to each of these protocol executions can be computed as a polynomial function of $x$ and $G'$ which we will denote as $g_i^m(x, G')$ and $g_i^u(x, G')$. The simulator can therefore use the subprotocol simulators as subroutines, producing the simulated transcript

$$\{ST^m(g_1^m(x, G'), h_1^m(G')), \ldots, ST^u(g_{2n}^u(x, G'), h_{2n}^u(G'))\} \tag{6.9}$$

where $ST^m$ and $ST^u$ denote the simulated transcripts of the minimum and union protocols, respectively.

We prove a hybrid argument over the simulated views for the minimum and set union protocols. First, define the hybrid distribution $H_i$ in which the first $i$ minimum/union protocols are simulated and the last $2n - i$ are real. Formally, let $H_i(x, y)$ denote the distribution:

$$\{ST^m(g_1^m(x, G'), h_1^m(G')), \ldots, ST^u(g_i^u(x, G'), h_i^u(G')),$$
$$RT^m(x_{i+1}, y_{i+1}), RT^u(x_{i+2}, y_{i+2}), \ldots, RT^u(x_{2n}, y_{2n})\}$$

We now prove that $H_0(x, y) \overset{c}{\equiv} H_{2n}(x, y)$ by showing that for all $i$, $H_i(x, y) \overset{c}{\equiv} H_{i+1}(x, y)$. For the sake of contradiction, assume the opposite, and choose $i$ so that $H_i(x, y) \overset{c}{\not\equiv} H_{i+1}(x, y)$. These two distributions differ in only one term, so there must be a polynomial-time distinguisher for either

$$ST^u(g_i^u(x, G'), h_i^u(G')) \text{ and } RT^u(x_i, y_i) \text{ or}$$
$$ST^m(g_i^m(x, G'), h_i^m(G')) \text{ and } RT^m(x_i, y_i)$$

However, this contradicts the privacy of the subprotocols, which implies that no such polynomial-time distinguishers exist. $\square$

We now show that for each execution of the set union and minimum subprotocols, $P_1$'s subprotocol input and the subprotocol output are computable as functions of $P_1$'s input and the output of the entire APSD protocol.

**Theorem 6.3.1.** *$m^{(k)}$ is efficiently computable as a function of $G'$.*

*Proof.* The edge weights found in $G'$ are $m^{(1)} < m^{(2)} < \ldots < m^{(n)}$. Therefore $m^{(k)}$ is the $k$th smallest edge weight in $G'$. $\square$

**Theorem 6.3.2.** *$S^{(k)}$ is efficiently computable as a function of $G'$.*

*Proof.* $S^{(k)}$ is the set of edges in $G'$ with weight $m^{(k)}$. □

**Theorem 6.3.3.** *$m_1^{(k)}$ is efficiently computable as a function of $G_1$ and $G'$.*

*Proof.* $m_1^{(k)}$ is the smallest edge weight in $G_1$ that is $> m^{(k-1)}$, allowing that $m^{(0)} = 0$. This is because all edges with weight $\leq m^{(k-1)}$ are in $R^{(k-1)}$. □

**Theorem 6.3.4.** *$S_1^{(k)}$ is efficiently computable as a function of $G_1$ and $G'$.*

*Proof.* $S_1^{(k)}$ is the set of edges in $G_1$ with weight $m^{(k)}$. □

### 6.3.2 All pairs shortest path

While there is only a single all pairs shortest distance solution for a given graph, there may be many all pairs shortest path solutions, because between a pair of points there may be many paths that achieve the shortest distance. As a side effect of engaging in the protocol described in section 6.3.1, the two participants learn an APSP solution. When defining the weight function $w_0^{(k)}$ by fixing violating triangles in $w_0'^{(k)}$ during step 7, a shortest path solution may be associated with the fixed edge. Specifically, if $w_0'^{(k)}(e_{ij}) + w_0'^{(k)}(e_{jk}) < w_0'^{(k)}(e_{ik})$, then the shortest path from $i$ to $k$ is through $j$.

In step 6 of subsequent iterations, when adding an edge $e_{ij} \in S^{(k)}$ to the set of blue edges, we can conclude that the shortest path from $i$ to $j$ is the edge $e_{ij}$ itself if $e_{ij} \notin S_0^{(k)}$, or is the shortest path solution as computed above if $e_{ij} \in S_0^{(k)}$.

Note that learning this APSP solution does not imply any violation of privacy, as it is the APSP solution implied by the APSD solution.

### 6.3.3 Single source shortest distance

The Single Source Shortest Distance (SSSD) problem is to find the shortest path distances from a source vertex $s$ to all other vertices [29]. An algorithm to solve APSD also provides the solution to SSSD, but leaks additional information beyond that of the SSSD solution and cannot be considered a private algorithm for SSSD. Therefore, this problem warrants its own investigation.

Similar to the protocol of section 6.3.1, the SSSD protocol on the minimum joint graph adds edges in order from smallest to largest. This protocol is very similar to Dijkstra's algorithm, but is modified to take two graphs as input.

1. Set $w_1^{(0)} = w_1$ and $w_2^{(0)} = w_2$. Color all edges incident on the source $s$ blue by putting all edges $e_{si}$ into the set $B^{(0)}$. Set the iteration count $k$ to 1.

2. Both parties privately compute the minimum length of blue edges in their graphs.

$$\begin{aligned} m_1^{(k)} &= \min_{e_{si} \in B^{(k-1)}} w_1^{(k-1)}(e_{si}), \\ m_2^{(k)} &= \min_{e_{si} \in B^{(k-1)}} w_2^{(k-1)}(e_{si}) \end{aligned}$$

3. Using the privacy-preserving minimum protocol, compute

$$m^{(k)} = \min(m_1^{(k)}, m_2^{(k)}).$$

4. Each party finds the set of blue edges in its graph with length $m^{(k)}$.

$$\begin{aligned} S_1^{(k)} &= \{e_{si} | w_1^{(k-1)}(e_{si}) = m^{(k)}\}, \text{ and} \\ S_2^{(k)} &= \{e_{si} | w_2^{(k-1)}(e_{si}) = m^{(k)}\} \end{aligned}$$

5. Using the privacy-preserving set union protocol, compute

$$S^{(k)} = S_1^{(k)} \cup S_2^{(k)}.$$

6. Color the edges in $S^{(k)}$ red by setting $B^k = B^{(k-1)} - S^{(k)}$. Define a weight function $w_1'^{(k)}$ by

$$w_1'^{(k)}(e) = \begin{cases} m^{(k)} & \text{if } e \in S^{(k)} \\ w_1^{(k-1)}(e) & \text{otherwise} \end{cases} \tag{6.10}$$

and a weight function $w_2'^{(k)}$ by

$$w_2'^{(k)}(e) = \begin{cases} m^{(k)} & \text{if } e \in S^{(k)} \\ w_2^{(k-1)}(e) & \text{otherwise} \end{cases} \tag{6.11}$$

7. Similar to the APSD algorithm, form the weight function $w_1^{(k)}$ by fixing the triangles in $w_1'^{(k)}$ that violate the triangle inequality and contain edges in $S^{(k)}$. $w_{2^{(k)}}$ is likewise formed from $w_2'^{(k)}$.

   If there are still blue edges remaining, go to step 2. Otherwise stop; both parties now have a graph with each edge incident on $s$ colored red, and with the weight of these edges equal to the shortest path distance from $s$ to each vertex.

### 6.3.4 Minimum spanning tree

Suppose that two frugal telephone companies wish to merge. Each company has a cost function for connecting any pair of houses, and they want to connect every house as cheaply as possible using the resources available to the merged company. In other words, they wish to compute $\mathrm{MST}(\mathrm{gmin}(G_1, G_2))$. If they can perform this computation privately, then both companies can see the final result without revealing their entire cost functions.

Both Kruskal's and Prim's algorithms for MST are easily turned into private protocols using our techniques, because the algorithms already consider edges in order from smallest to largest. At each iteration, Kruskal's algorithm adds the shortest edge such that its addition does not form a loop. It is a simple task for each party to compute the set of edges which would not form loops, and then to privately compute the length of the shortest edge in this set. One problem arises when there are multiple edges that share this length. In the shortest path algorithms, we addressed this issue by adding all edges of appropriate length at the same time using the private set union protocol, but this will not work for MST. Instead, we can assign a canonical ordering to the edges, and at each step find the shortest length edges that are canonically "first." This will allow a simulator to determine, given the final MST, in what order the edges arrived.

## 6.4 Complexity Analysis

For each algorithm considered in this chapter, we calculate the number of rounds, the total communication complexity, and the computational complexity, and compare them with the generic method. Using Yao's method on a circuit with $m$ gates and $n$ inputs requires $O(1)$ rounds, $O(m)$ communication, and $O(m + n)$ computational overhead. Lindell and Pinkas note in [78] that the computational overhead of the $n$ oblivious transfers in each invocation of Yao's protocol typically dominates the computational overhead for the $m$ gates, but for correct asymptotic analysis we must still consider the gates.

**Complexity of privacy-preserving APSD.** For our analysis we will assume that the edge set $E$ has size $n$, and that the maximum edge length is $l$. The generic approach to this problem would be to apply Yao's Method to a circuit that takes as input the length of every edge in $G_1$ and $G_2$, and returns as output $G = \text{APSD}(\text{gmin}(G_1, G_2))$. Clearly, such a circuit will have $2n \log l$ input bits. To count the number of gates, note that a circuit to implement Floyd-Warshall requires $O(n^{3/2})$ minimums and $O(n^{3/2})$ additions. For integers represented with $\log l$ bits, both of these functionalities require $\log l$ gates, so we conclude that Floyd-Warshall requires $O(n^{3/2} \log l)$ gates. To compute gmin requires $O(n \log l)$ gates, but this term is dominated by the gate requirement for Floyd-Warshall. We conclude that the generic approach requires $O(1)$ rounds, $O(n^{3/2} \log l)$ communication, and $O(n^{3/2} \log l)$ computational overhead.

The complexity of our approach depends on the number of protocol iterations $k$, which is equal to the number of different edge lengths that appear in the solution graph. In iteration $i$, we take the minimum of two $(\lg l)$-bit integers, and compute a set union of size $s_i$. Because each edge in the graph appears in exactly one of the set unions, we also know that $\sum_{i=1}^{k} s_i = n$.

First we will determine the contribution to the total complexity made by the integer minimum calculations. If we use Yao's protocol, then each integer minimum requires a constant number of communication rounds, $O(\lg l)$ inputs, and $O(\lg l)$ gates, so the $k$ calculations together contribute $O(k)$ rounds, $O(k \lg l)$ communication complexity, and $O(k \lg l)$ computational complexity.

Complexity contribution of the set union subprotocols depends on whether we use the iterative method or the tree pruning method as described in section 6.2. If the iterative method is used, then the $k$ invocations of set union require a total of $O(n)$ rounds, $O(k \lg n)$ communication complexity, and $O(k \lg n)$ computational complexity. If the tree-pruning method is used, then $O(k \lg n)$ rounds are required, but the communication and computational complexity remains the same. The asymptotically better performance of the iterative method hides the fact that each of the $k$ rounds requires $O(\lg n)$ oblivious transfers, which are considerably more expensive than the $O(|s_i|)$ private Bit-Or computations performed in each of the $\lg u$ rounds of the tree-pruning method.

Using the iterative method for set union, and noting that $k = O(n)$, we conclude that our APSD protocol requires $O(n)$ communication rounds,

$O(n \log n + n \log l)$ communication complexity, and $O(n \log n + n \log l)$ computational complexity. As compared to the generic approach, we have traded more rounds for better overall complexity.

**Complexity of privacy-preserving SSSD.** Complexity of SSSD is similar to that of APSD, except that the number of rounds is $k = O(v)$ and the total number of set union operations is $v$, where $v$ is the number of vertices ($O(e^{1/2})$). We conclude that our protocol requires $O(v)$ rounds, $O(v(\log v + \log l))$ oblivious transfers, and $O(v(\log v + \log e))$ gates. A generic solution, on the other hand, would require $O(v^2 \log l)$ oblivious transfers.

## 6.5 Proof of Private APSD Protocol Correctness

Before proving the algorithm correct, we prove some supporting lemmas.

**Lemma 6.5.1.** *If an edge $e \in R^k$ and $w_0^{(k)}(e) = l$ then $\forall j > k, w_0^{(j)}(e) = l$.*

*Proof.* Intuitively, this says that once the protocol establishes the length of a red edge, it never changes. This follows from the protocol lacking operations that alter the length of red edges. □

**Lemma 6.5.2.** *For an edge $e \in R^{(k)}$, $w_0^{(k)}(e) \leq m^{(k)}$*

*Proof.* In step 6 of iteration $k$, for edges $e \in S^{(k)}$ we set $w_0^{(k)}(e) = m^{(k)}$ and $e \in R^{(k)}$. Apply lemma 6.5.1 to complete the proof. □

**Lemma 6.5.3.** *For an edge $e \in B^{(k)}$, $w_0^{(k)}(e) > m^{(k)}$*

*Proof.* First, we show that for an edge $e \in B^{(k)}$, $w_0'^{(k)}(e) > m^{(k)}$. If $w_0'^{(k)}(e) = m^{(k)}$ then $e \in S^{(k)}$ (and $e \notin B^{(k)}$). If $w_0'^{(k)}(e) < m^{(k)}$ and $e \in B^{(k)}$, then $w_0^{(k-1)}(e) < m^{(k)}$ and we would have defined a smaller $m^{(k)}$.

Now, for those edges $e$ where we have $w_0^{(k)}(e) < w_0'^{(k)}(e)$ because of step 7, we still have $w_0^{(k)}(e) > m^{(k)}$ because the right-hand side of the assignment is strictly greater than $m^{(k)}$. $\square$

**Lemma 6.5.4.** *For all edges $e$, $e \in R^{(k)} \leftrightarrow w_0^{(k)}(e) \leq m^{(k)}$ and $e \in B^{(k)} \leftrightarrow w_0^{(k)}(e) > m^{(k)}$*

*Proof.* This is an immediate consequence of lemmas 6.5.2 and 6.5.3. $\square$

**Lemma 6.5.5.** *For every red edge $e_{ij} \in R^{(k)}$, $w_0^{(k)}(e_{ij}) = d_G(i,j)$.*

*Proof.* The proof is by induction on $k$. For $k = 0$, the result is trivial. We will now assume that the result holds for values less than $k$ and prove it for $k$.

Because of lemma 6.5.1, it is sufficient to prove that for edges $e_{ij} \in S^{(k)}$, $d_G(i,j) = m^{(k)}$. We consider two cases.

1. The shortest path from $i$ to $j$ in $G$ is the edge $e_{ij}$.
   In this case, $d_G(i,j) = \min(w_1(e_{ij}), w_2(e_{ij}))$. To complete the proof, it's enough to show that $w_0^{(k-1)}(e_{ij}) \geq d_G(i,j)$. Suppose that in some iteration $h < k$ we set $w_0^{(h)}(e_{ij}) = w_0'^{(h)}(e_{ik}) + w_0'^{(h)}(e_{kj})$ in step 7. Then

by inductive hypothesis, this implies a shorter path from $i$ to $j$ than the edge $e_{ij}$ which is a contradiction.

2. The shortest path from $i$ to $j$ in $G$ is through $k$.

   In this case, $d_G(i,j) = d_G(i,k) + d_G(k,j)$. WLOG, assume that $w_0^{(k)}(e_{ik}) \geq w_0^{(k)}(e_{kj})$. Then by lemmas 6.5.1 and 6.5.4, we have that for some $h < k$, $w_0^{(k)}(e_{ik}) = m^{(h)}$. This means that in step 7 of iteration $h$ the protocol set $w_0^{(h)}(e_{ij}) = w_0^{(h)}(e_{ik}) + w_0^{(h)}(e_{kj})$. By the inductive hypothesis, $w_0^{(h)}(e_{ik}) = d_G(i,k)$ and $w_0^{(h)}(e_{kj}) = d_G(k,j)$. We conclude that $w_0^{(h)}(e_{ij}) = d_G(i,k) + d_G(k,j)$ and therefore that $w_0^{(k)}(e_{ij}) \leq d_G(i,k) + d_G(k,j)$. By the same argument as in the first case, we also have $w_0^{(k)}(e_{ij}) \geq d_G(i,k) + d_G(k,j)$. Therefore, $m^{(k)} = d_G(i,k) + d_G(k,j) = d_G(i,j)$.

□

It is now a simple task to prove algorithm correctness.

*Correctness.* Suppose the algorithm terminates after $n$ iterations. Then $R^{(n)} = E$. Apply lemma 6.5.5. □

## 6.6 Conclusions

In this chapter, we presented privacy-preserving protocols that enable two honest but curious parties to compute APSD and SSSD on their *joint graph.* A related problem is how to construct privacy-preserving protocols for graph *comparison*. Many of these problems (*e.g.*, comparison of the graphs' respective maximum flow values) reduce to the problem of privacy-preserving comparison of two values, and thus have reasonably efficient generic solutions. For other problems, such as graph isomorphism, there are no known polynomial-time algorithms even if privacy is not a concern. Investigation of other interesting graph algorithms that can be computed in a privacy-preserving manner is a topic of future research.

# Chapter 7

# Privacy-Preserving Remote Diagnostics

## 7.1 Introduction

Diagnostic programs, typically represented as decision trees or binary branching programs, are the cornerstone of expert systems and data analysis tools. Learning and evaluating diagnostic programs which classify data on the basis of certain features are among the most fundamental data mining tasks.

Evaluation of a diagnostic program on a remote user's data often presents privacy risks to both the user and the program's owner. The program's owner may not want the user to learn the entire contents of the diagnostic program, while the user may not want to reveal his local data to the program's owner. For example, consider a medical expert system, where the diagnostic program is the realization of a substantial investment, and the data on which the program is evaluated contain information about the user's health.

Another example is remote software fault diagnosis, which is an increasingly popular support method for complex applications. The details of remote software diagnostic systems differ (see Section 7.3), but there are many commonalities. An application does something undesirable (crashes, becomes slow or unresponsive, quits with an obscure error message), and the runtime system gathers some data about the problem. The software manufacturer uses this information to diagnose the problem, usually by reading a small subset of the data. Users are typically required to ship all fault-related data to the manufacturer. For example, most users of Microsoft Windows have encountered the (in)famous "send error report" button.

The data gathered by the runtime system may contain sensitive information, such as passwords and snippets of the user's documents. Many users are not willing to reveal this information to the software manufacturer. On the other hand, software manufacturers often view their proprietary diagnostic programs as valuable intellectual property. Diagnostic programs may reveal the application's support history, unpatched security vulnerabilities, and other information about the implementation and internal structure of the application that the manufacturer may prefer to keep secret.

This chapter describes a method for *privacy-preserving evaluation of diagnostic branching programs.* This problem is different from privacy-preserving *learning* of decision trees, which is considered in Chapter 8 of this thesis. We assume that the diagnostic program already exists, in the form of a binary decision tree or branching program, and investigate how to *apply* it to the user's data in such a way that the program is not revealed to the user, and the user's data are not revealed to the program's owner.

**Our contributions.** We present a practical, provably secure interactive protocol for privacy-preserving evaluation of branching programs. The protocol takes place between a Server, in possession of a binary branching program $T$, and a User, in possession of an attribute vector $v$. The User learns $c = T(v)$, the diagnostic label that $T$ assigns to $v$. The Server may or may not learn the label—we consider both variants.

Our protocol does not reveal any useful information except the outcome of the computation, which is the diagnostic label in this case. In particular, the User does not learn *how* the branching program arrived at the diagnosis, nor which of the User's attributes it considered, nor the topology of the branching program, nor any other diagnostic labels that it may contain. The Server, on the other hand, learns nothing whatsoever about the User's local data.

We emphasize the strong privacy properties achieved by our protocol. For example, secrecy of the program being evaluated is *not* the standard requirement of secure multi-party computation, which usually assumes that the program is public, and only the parties' respective inputs are secret. In many of our applications, the user should not learn *which* of his attributes are considered by the branching program. If the attribute vector is very large (as is the case, for example, in software fault diagnostics, where the attribute vector is a record of the user's runtime environment), achieving these security properties efficiently is a difficult challenge.

Our branching program evaluation protocol combines in a novel way several cryptographic techniques such as homomorphic encryption, blinding, and Yao's "garbled circuits" method. Yao's method is used in a somewhat unusual way, not simply as a black-box realization of secure circuit evaluation. We exploit the details of circuit representation in Yao's protocol to implement a *conditional oblivious transfer* primitive needed by our protocol.

We present a substantial case study, in which we use our method to implement a privacy-preserving version of Clarify [60], a system for remote diagnosis of software faults. We apply our protocol to the decision trees generated by Clarify for several large, real-world applications such as `gcc` and `latex`, and demonstrate that its performance is efficient for many practical scenarios.

While there have been many theoretical results in the field of secure multi-party computation, actual implementations and working systems are extremely rare. Experimental evaluation of our prototype implementation demonstrates that our protocol performs significantly better than the generic methods.

## 7.2 Secure Evaluation of Branching Programs

We now describe our protocol for the secure evaluation of binary branching programs. The protocol is executed between a Server, in possession of a branching program (formally defined in Section 7.2.1), and a User, in possession of an attribute vector. Let $k$ be the number of nodes in the branching program, and $n$ be the number of attributes.

In most practical scenarios, $n$ is significantly larger than $k$; our protocol is optimized for this case. In particular, the size of the securely transformed branching program is independent of $n$.

### 7.2.1 Branching programs

In this section, we formally define branching programs, which include binary classification or decision trees as a special case. Let $\mathcal{V} = v_1, \ldots, v_n$ be the vector of User's attributes. Each attribute value is an $\ell$-bit integer. (In our experiments, $\ell = 32$, which appears to be sufficient for most practical scenarios.)

A binary branching program $T$ is a triple $\langle \{P_1, \ldots, P_k\}, L, R \rangle$. The first element is a set of nodes. For $i \leq l$, $P_i$ are *decision nodes.* For $i > l$, $P_i$ are *classification nodes.*

Decision nodes are the internal nodes of the program. Each decision node is a pair $\langle t_i, \alpha_i \rangle$, where $\alpha_i$ is the index of an attribute, and $t_i$ is the threshold value with which $v_{\alpha_i}$ is compared in this node. The same value of $\alpha$ may occur in many nodes, *i.e.*, the same attribute may be evaluated more than once. For each decision node $i$, $L(i)$ is the index of the next node if $v_{\alpha_i} \leq t_i$; $R(i)$ is the index of the next node if $v_{\alpha_i} > t_i$. Functions $L$ and $R$ are such that the resulting directed graph is acyclic.

Classification or diagnosis nodes are the leaf nodes of the program. Each leaf node consists of a single classification label $\langle d_i \rangle$.

To evaluate the branching program on some attribute vector $\mathcal{V}$, start at $P_1$. If $v_{\alpha_1} \leq t_1$, set $h = L(1)$, else $h = R(1)$. Repeat the process recursively for $P_h$, and so on, until reaching one of the leaf nodes and obtaining the classification.

### 7.2.2 Security requirements

The objective of our protocol is to *securely* evaluate $T$ on $\mathcal{V}$. The protocol should reveal nothing to the Server. The User should learn $T(\mathcal{V})$, which is a classification label contained in one of the leaves of the branching program $T$. The User is also permitted to learn the total number of nodes of $T$ (see the discussion in Section 7.2.6) and the length of the path from the root node of $T$ to the leaf containing the result of evaluation, *i.e.*, the label assigned by $T$ to $\mathcal{V}$.

The User should not learn anything else about $T$. In particular, the User should not learn *which* attributes from $\mathcal{V}$ have been considered by $T$, with what threshold values they have been compared, the outcome of any comparison, and so on.

The requirement that attribute selection be oblivious precludes a naïve application of secure multi-party computation (SMC) techniques. In standard SMC, each participant knows which of his inputs have been used in the computation. While it is possible to create a circuit that takes all of the User's attributes as inputs and ignores those not used by $T$, this circuit would be impractically large ($\mathcal{V}$ may contains tens of thousands of attributes). A detailed discussion can be found in Section 7.2.5.

### 7.2.3 Privacy-preserving offset integer comparison

Our protocol for the evaluation of branching programs requires a secure sub-protocol for the comparison of integer values. The Privacy-Preserving Offset Integer Comparison protocol takes place between two parties, Alice and Bob. Bob has an $\ell$-bit integer $x$, while Alice has $\ell$-bit integers $b$ and $t$, and output keys $h^0$ and $h^1$. At the end of the protocol execution, Bob learns $h^0$ if $x - b \mod 2^\ell < t$ and $h^1$ otherwise, while Alice learns nothing. We denote this functionality as $\textsc{Compare}(x, b, t, h^0, h^1)$. Note that in the special case where $b = 0$, and $h^0$ and $h^1$ are the single bits 0 and 1, this is the same as Yao's millionaires' problem.

This problem is also known as *conditional oblivious transfer* with a "greater than" predicate [13]. We assume that participants are computationally bounded, since the encryption scheme used in our protocol is only computationally secure. In this case, Yao's method is the most efficient currently known approach [13].

Yao's garbled circuits provide a relatively efficient protocol for integer comparison because the circuit needed to compare two $\ell$-bit integers is relatively small. As described in Section 2.5.1, in our protocol the circuit creator does not provide the mappings from output-wire keys to actual output-wire bits, so that Bob learns one of the two keys, but not the actual result of the comparison.

### 7.2.4 Secure branching program protocol

The protocol runs in three phases.

**Phase I (offline): Creation of the secure branching program.** This is an offline pre-computation executed by the Server. Using Algorithm 3, the Server converts the original branching program $T$ into its secure equivalent $T'$. Algorithm 3 does not require any interaction with the User or knowledge of the User's identity. For example, the Server may maintain a large store of secure branching programs (all representing differently randomized transformations of the same $T$), which is replenished during idle periods when the Server's machines have many spare cycles.

Algorithm 3 converts the nodes in the branching program $T$ into secure nodes in the branching program $T'$. Each classification node is replaced by an encryption of its classification label so that its contents will remain unknown to the User unless the appropriate decryption key is obtained. Each decision node is replaced by a small garbled circuit implementing offset integer comparison (see Section 7.2.3). This circuit enables the User to learn one of two keys, depending on the comparison between the User's attribute value (offset by a blinding value) and the decision node's threshold value. The revealed key decrypts the next node on the evaluation path.

Because the User should not know *which* attribute is being compared to a threshold, the User's input to the garbled circuit is *blinded* by the Server (in phase II, described below) by adding a random $(\ell + \ell')$-bit value that the

User does not know. Here $\ell'$ is the statistical security parameter, set to 80 bits in our implementation. The blinding values $b_1, \ldots, b_k$ are generated randomly by the Server in Phase I. They will be subtracted from the User's input to the circuit before it is compared to the threshold.

**Phase II: Oblivious attribute selection.** In this phase, the User obtains the blinded attribute values which will be used as inputs to the COMPARE circuits in the secure decision nodes created in Phase I. First, the User creates an instance of the additively homomorphic public-key encryption scheme, and encrypts each attribute in his attribute vector with the public key $y$ (this can take place offline). The User sends the entire encrypted attribute vector to the Server along with the public key $y$.

For node $i$, the blinding value chosen in Phase I is $b_i$, and the attribute to be compared is $\alpha_i$. Thus, the User needs to learn $v_{\alpha_i} + b_i$. The Server cannot compute this value directly without learning $v_{\alpha_i}$ (which violates the User's privacy), but he can compute $\{v_{\alpha_i} + b_i\}_y$ since he was provided with $\{v_{\alpha_i}\}_y$, he can independently compute $\{b_i\}_y$, and the encryption is homomorphic. He computes this encrypted value and sends it to the User.

The random blinding value $b_i$ added by the Server to the encrypted $\ell$-bit attribute $v_{\alpha_i}$ is $\ell'$ bits longer than $v_{\alpha_i}$. Therefore, it statistically hides $v_{\alpha_i}$ (and thus does not reveal which attribute the Server chose) when $\ell'$ is sufficiently large (80 bits in our implementation). Note that $2^{\ell+\ell'}$ is much smaller than the order of the group in which plaintext addition is done under encryption.

**Input:** Branching program $T = \langle\{P_1, \ldots, P_k\}, L, R\rangle$ (see Section 7.2.1). For $i \leq l$, $P_i$ is a decision node $\langle t_i, \alpha_i\rangle$. For $i > l$, $P_i$ is a classification node containing label $\langle d_i\rangle$.
**Outputs:**
(i) Secure branching program $T'$
(ii) $k$ random $\ell + \ell'$-bit blinding values $b_1, \ldots, b_k$
(iii) $2 \cdot k \cdot \ell$ random wire keys $w^0_{ij}, w^1_{ij}$ for $1 \leq i \leq k, 1 \leq j \leq \ell$

```
CREATESECUREPROGRAM
 1: let $Q$ be a random permutation of the set $1, ..., k$ with $Q[1] = 1$
 2: Generate random keys $\kappa_1, ..., \kappa_k$ to be used for encrypting the decision
    nodes.
 3: for $i = 1$ to $k$ do
 4:    Generate $2 \cdot \ell$ random wire keys $w^0_{ij}, w^1_{ij}$ for $1 \leq j \leq \ell$ (to be used for
       encoding the User's input into the garbled threshold comparison circuit).
 5:    Generate a random $\ell + \ell'$-bit blinding value $b_i$; store $b_i$ and $b'_i = b_i$
       mod $2^\ell$.
 6:    let $\tilde{i} = Q[i]$
 7:    if $P_i$ is a classification node $\langle d_i\rangle$ then
 8:       let $S_{\tilde{i}} = \{\text{"label"}, d_i\}_{\kappa_{\tilde{i}}}$, where $\{y\}_\kappa$ is the encryption of $y$ under key
          $\kappa$ using a semantically secure symmetric-key encryption scheme. (We
          assume that all plaintexts are padded so that the ciphertexts of deci-
          sion nodes and classification nodes have the same size.)
 9:    else if $P_i$ is a decision node $\langle t_i, \alpha_i\rangle$ then
10:       Use the subroutine YAO for generating garbled circuits (see Sec-
          tion 2.5.1) to generate a secure circuit $C_i$ for the offset integer com-
          parison functionality (see Section 7.2.3) COMPARE$(x, b'_{\tilde{i}}, t_i, \mathbf{L}, \mathbf{R})$ =
          if $x - b'_{\tilde{i}} \mod 2^\ell < t_i$ then return $\mathbf{L}$ else return $\mathbf{R}$
          where $\mathbf{L} = (Q[L(i)], \kappa_{Q[L(i)]})$,
                $\mathbf{R} = (Q[R(i)], \kappa_{Q[R(i)]}))$
          Use $w^0_{\tilde{i}j}, w^1_{\tilde{i}j}$ $(1 \leq j \leq \ell)$ to encode, respectively, 0 and 1 on the $\ell$ wires
          corresponding to input $x$.
11:       let $S_{\tilde{i}} = \{C_i\}_{\kappa_{\tilde{i}}}$
12:    end if
13: end for
14: return $T' = \langle\{S_1, ..., S_k\}, \kappa_1\rangle$
```

**Algorithm 3:** Convert a branching program into a secure branching program

The User uses his private key to decrypt $v_{\alpha_i} + b_i$. By taking $v_{\alpha_i} + b_i \mod 2^\ell$, the User obtains $s_i$, his $\ell$-bit input into the garbled offset integer comparison circuit.

Next, the User acts as the chooser in $\ell$ instances of 1-out-of-2 oblivious transfer with the Server to learn the garbled wire keys corresponding to his input value $s_i$. Note that this does not reveal $s_i$ to the Server. Now the User has all the wire key values he needs to evaluate $T'$ in phase III.

**Phase III: Evaluation of the secure branching program.** In the last phase, the User receives the secure branching program $T'$ from the Server along with $\kappa_1$, and evaluates it locally by applying Algorithm 5 on inputs $(T', 1, \kappa_1)$.

Evaluation does not reveal anything to the User except the label at the end of the evaluation path. At each step, the User applies one of the comparison circuits $C_h$ to the value $s_h$ (encoded as a set of wire keys—see Section 2.5.1), but he does not know which of his attributes is hidden in $s_h$. The User thus learns the index of the next node and the decryption key, but not the result of the comparison.

The only information leaked by the evaluation procedure is (i) the total number of nodes in the program $T'$, (ii) the number of nodes that have been evaluated before reaching a classification node (note that in a full decision tree this number does not depend on the path taken), and (iii) the classification label $d$.

If the usage scenario requires the Server to learn the classification label,

**User's input:** Attribute vector $v_1, \ldots, v_n$ with $\ell$-bit attribute values
**Server's input:** For each node $t_i$ of $T'$, $\alpha_i$ is the index of the User's attribute which is being compared in this node (if $t_i$ is not a decision node, $\alpha_i$ is chosen randomly); $b_i$ is the random $(\ell+\ell')$-bit value generated as part of CreateSecureProgram.
**Outputs for the User:**
(i) $s_1, \ldots, s_k$ where $\forall i\ s_i = v_{\alpha_i} + b_i \mod 2^\ell$
(ii) For each $i$, wire keys $w_{i1}, \ldots, w_{i\ell}$ encoding $s_i = v_{\alpha_i} + b'_i \mod 2^\ell$ on the input wires of circuit $C_i$ (see Algorithm 3).
**Output for the Server:** $\perp$

```
ObliviousAttributeSelection
1: The User generates a public/private key pair $(y,x)$ of a homomorphic encryption scheme, and sends the public key $y$ to the Server.
2: for $i = 1$ to $n$ do
3:     The User sends $\{v_i\}_y$ to the Server.
4: end for
5: for $i = 1$ to $k$ do
6:     Server computes $\{v_{\alpha_i} + b_i\}_y$ from $\{v_{\alpha_i}\}_y$ and $b_i$ using the homomorphic property of the encryption scheme, and sends this value to the User.
7:     The User decrypts to find $v_{\alpha_i} + b_i$ and then computes $s_i = v_{\alpha_i} + b_i \mod 2^\ell = v_{\alpha_i} + b'_i \mod 2^\ell$
8:     for $j = 1$ to $\ell$ do
9:         The Server and the User execute $OT_2^1$ oblivious transfer protocol. The User acts as the chooser; his input is $s_i[j]$, i.e., the $j$th bit of $s_i$. The Server acts as the sender; his inputs are wire keys $w_{ij}^0$ and $w_{ij}^1$, encoding, respectively, 0 and 1 on the $j$th input wire of threshold comparison circuit $C_i$ (see Algorithm 3).
10:    end for
11:    As the result of $\ell$ oblivious transfers, the User learns wire keys $w_{i1}, \ldots, w_{i\ell}$ encoding his input $s_i$ into the circuit $C_i$. Note that the User cannot yet evaluate $C_i$ because he does not know the key $\kappa_i$ under which $C_i$ is encrypted.
12: end for
```

**Algorithm 4:** Oblivious attribute selection

too, the User simply sends $d$ to the Server. If the Server should learn the classification label and the User should learn nothing, then the Server can replace the labels with ciphertexts encrypting the labels under the Server's public key; when the User obtains a ciphertext at the end of evaluation, he sends it to the Server.

```
Inputs: Secure program $T'$, node index $h$ with corresponding node encryption key $\kappa_h$, and, for each $i$ such that $1 \leq i \leq k$, wire keys $w_{i1}, \ldots, w_{i\ell}$.
Output: Classification label $c$ such that $c = T(\mathcal{V})$

EvaluateSecureProgram($T', h, \kappa_h$)
 1: Use key $\kappa_h$ to decrypt node $S_h$ of $T'$ and obtain $C_h$.
 2: if $C_h = \langle$"label"$, d\rangle$ then
 3:     $C_h$ is a classification node.
        return label $d$.
 4: else if $C_h$ is a garbled circuit then
 5:     Evaluate $C_h$ on inputs $w_{h1}, \ldots, w_{h\ell}$.
 6:     As the result of evaluation, obtain the pair $(h', \kappa_{h'})$ encoding the output
        wire value.
 7:     return EvaluateSecureProgram($h', \kappa_{h'}$).
 8: else
 9:     error "Secure program is not properly formed!"
10: end if
```

**Algorithm 5:** Evaluation of secure branching program

We emphasize that the User cannot simply re-run the program evaluation algorithm of Phase III on the same secure program $T'$ and a *different* attribute vector, thus learning more about the original branching program. After learning the wire keys corresponding to his (blinded) attributes during Phase II, the User can evaluate only a *single* path in the branching program—that corresponding to the attribute vector he used as his input into the protocol.

There is no way for the User to learn the random wire keys encoding other possible inputs to the program.

In order to evaluate $T$ on a different attribute vector, the User must re-run the entire protocol starting from Phase I. He will then obtain a different secure program $T''$ and a different set of wire keys. Our protocol maintains the invariant that, for every secure branching program, there is only one path that can be evaluated by the User, and this path appears random to the User.

### 7.2.5 Efficiency and comparison with generic techniques

In the secure branching program created by Algorithm 3, each of the decision nodes of the original program is replaced by a garbled Yao circuit for comparing two (offset) $\ell$-bit integers. Each such circuit requires $\log \ell$ gates, for a total of $k \cdot \log \ell$ gates (this is a conservative estimate, since some of the nodes are classification nodes). Note that the size of the circuit is independent of the number of the User's attributes $n$. Algorithm 4 requires $k \cdot \ell$ $OT_2^1$ oblivious transfers to transfer the wire keys corresponding to the User's (blinded) inputs into each of the $k$ nodes.

An alternative to using our protocol from Section 7.2.4 is to use generic techniques that enable secure computation of any two-party functionality, represented either as a boolean circuit [80, 132] or a binary decision diagram [53] (the latter may be a better choice for branching diagnostic programs).

A naïve way to implement the secure evaluation of binary branching programs using generic techniques would be to have the Server take his specific branching program, transform it into an equivalent secure program using, say, the standard garbled circuit techniques (see Section 2.5.1), and have the User evaluate the garbled circuit on his attribute vector.

This does *not* satisfy our security requirements. First of all, the topology of the program is revealed to the User. In generic secure multi-party computation (SMC), it is usually assumed that the function to be computed is known to both parties. Yao's garbled circuit technique works even if the circuit evaluator does not know the truth tables associated with the individual gates, but it reveals the topology of the circuit being evaluated. By contrast, our protocol only reveals the length of the evaluation path and the total number of nodes; it leaks no other information about the rest of the branching program. Even worse, with the naïve approach the User learns on *which* of his attributes the program was evaluated, thus violating one of our core security requirements (see Section 7.2.2).

To ensure obliviousness of the User's input selection, the SMC functionality must be defined so that it takes *any* branching program of a given size (as opposed to the specific Server's program) and securely applies it to *any* attribute vector of a given length.

We have attempted to implement such a functionality using the Fairplay compiler [82], which converts any two-party functionality into an equivalent garbled circuit. Unfortunately, the Fairplay compiler is memory-bound, and in our experiments it was unable to compile functionalities that would allow us to apply branching programs of realistic size to realistic attribute vectors.

|  |  | Server |  | User |  |
|---|---|---|---|---|---|
| Nodes | Attrib. | Computation | Communication | Computation | Communication |
| 15 | 100 | *67* vs. **2** sec | *1,292* vs. **263** KB | *76* vs. **3** sec | *528* vs. **98** KB |
| 63 | 100 | *72* vs. **7** sec | *1,799* vs. **1121** KB | *79* vs. **14** sec | *528* vs. **351** KB |
| 15 | 1000 | *605* vs. **2** sec | *11,388* vs. **263** KB | *706* vs. **4** sec | *5,277* vs. **255** KB |
| 63 | 1000 | $X$ vs. **7** sec | $X$ vs. **1121** KB | $X$ vs. **12** sec | $X$ vs. **508** KB |
| *Cursive* - Fairplay, **Bold** - our protocol, $X$ - failed to compile |  |  |  |  |  |

Table 7.1: Comparison of protocols for the evaluation of branching programs

On a machine with 4 Gigabytes of RAM, the compiler runs out of memory when attempting to compile the functionality that applies a 63-node branching program to a 400-attribute vector.

Table 7.1 gives comparative measurements of online computation and communication for a few sample configurations. Our experimental setup, along with the detailed performance analysis of our protocol, can be found in Section 7.4.

Some of the negative aspects of Fairplay, such as running out of memory even on relatively small configurations, may be due to the particular compiler implementation rather than the inherent flaws of the generic approach. Nevertheless, our protocol described in Section 7.2.4 provides a superior solution for the specific task of secure branching program evaluation.

### 7.2.6 Achieving complete privacy

The protocol of Section 7.2.4 reveals the total number of nodes in the branching program and the length of the evaluation path corresponding to the User's attribute vector. This information appears harmless in practice, but, if necessary, it can be hidden, provided that there exist upper bounds $B$ on the number of nodes and $P$ on the length of the longest evaluation path.

To hide the size of the branching program, the Server can create $B - k$ random ciphertexts (which will never be decrypted by the User), and mix them randomly with the real encrypted nodes of the secure program $T'$. Semantic security of the encryption scheme used to encrypt individual nodes guarantees that the User cannot tell the difference between an encryption of a real node that he did not reach in his evaluation, and a random ciphertext of the same size. When padded in this way, secure versions of *all* branching programs will contain exactly $B$ ciphertexts.

To hide the length of evaluation paths, first transform the branching program into a decision tree, so that each node has a fixed depth. Then transform it into a full tree of depth $P$ by replacing classification nodes at depth $p < P$ with full trees of depth $(P - p + 1)$, in which every leaf contains the original classification. In the resulting tree, every evaluation path has length $P$.

## 7.3 Remote Software Diagnostics

In this section, we give a brief introduction to the problem of remote software fault diagnostics, and then use our protocol of Section 7.2 to implement a privacy-preserving version of Clarify, a practical system for software fault diagnosis [60].

Microsoft error reporting is an example of a remote software diagnosis tool [88]. A Microsoft error report has two purposes. The first purpose is to gather extensive information about a software failure to enable Microsoft engineers to fix the software problem. This chapter focuses on the problem of privately *evaluating* decision trees, while Chapter 8 focuses on the problem of privately *learning* decision trees.

The second purpose is to improve the user's experience by providing a message describing the user's problem and how the user can avoid the problem in the future. Our case study addresses the second purpose, where a server provides feedback to the user about the user's problem. Windows Vista includes a prominent item on the control panel called "Problem reports and solutions" [87] that allows users to get the latest information about a particular software problem from Microsoft's web site. The Ubuntu Linux distribution also contains new features to generate more information about software failures to help users [122].

Microsoft's privacy statement about the information it collects for software problem diagnosis [86] acknowledges that problem reports can compromise users' privacy. Problem reports contain the contents of memory for the program that failed, and this memory "might include your name, part of a document you were working on or data that you recently submitted to a website." The policy says that users concerned about the release of personal or confidential information should *not* send problem reports. Of course, users who do not send problem reports cannot benefit from remote fault diagnostics. Corporate users in particular have expressed concern that remote diagnostics could reveal their intellectual property [92]. Ubuntu's documentation also acknowledges security and privacy risks associated with data-rich fault reports.

The protocol presented in this chapter enables the user to obtain a support message in a privacy-preserving fashion. The user does not reveal anything to the software manufacturer about his or her local data, and the software manufacturer does not reveal to the user how the user's local data was mapped to a diagnostic message.

**Privacy of diagnostic programs.** It may appear that the software manufacturer should simply send the diagnostic program to the user or, better yet, integrate it directly into the supported application. Many software manufacturers, however, view their diagnostic programs as valuable intellectual property. They state this explicitly in the legal documents that accompany the diagnostic software [51] and sue competitors who obtain access to their diagnostic programs [100]. Updating widely deployed software with new support messages and diagnostic tools is not always feasible, either, since many users simply don't install patches.

Moreover, diagnostic programs can reveal vulnerabilities in deployed software. For example, a single message from Microsoft's Dr. Watson diagnostic tool was sufficient to reveal to any user who experienced a particular fault an exploitable buffer overflow [84] (had the entire Dr. Watson diagnostic tree been shipped to every Windows user instead of being evaluated on Microsoft's servers, even users who did not experience the fault could discover the vulnerability by analyzing the diagnostic tree). In the diagnostic trees produced by the Clarify toolkit for `gzprintf`, which is one of our benchmarks, the inner nodes of the diagnostic tree directly point to the function that contains a security vulnerability. With a lively, semi-legal market in information about software vulnerabilities [109], software manufacturers have a strong disincentive to completely reveal all known faults and bugs in their applications.

We emphasize that we do not promote "security by obscurity." Software manufacturers should patch the bugs and vulnerabilities in their programs as soon as practicable. From a purely pragmatic perspective, however, they should not be forced to choose between not providing diagnostic support, or else revealing every internal detail of their applications and diagnostic tools. In reality, when faced with such a stark choice, many will decide to not provide support at all, resulting in a poorer experience for the users who are not willing to disclose their own local data.

**Runtime data collection and fault diagnosis.** Typically, the runtime environment records some abstraction of the program's behavior. Different abstractions have different cost and accuracy tradeoffs (*e.g.*, see [32, 60]). For instance, one abstraction counts how many times each function in the program was called, another counts function call sites that satisfy a certain predicate (*e.g.*, equal to zero) on the function's return value, and so on.

For the purposes of this chapter, we abstract from the details of the program behavior "dumps" generated by the runtime environment, and refer to individual data items simply as *attributes*. We assume that the vector of attributes has a fixed maximum size, which can be quite large—for example, a vector of function callsite counters may include dozens of thousands of attributes. Note that the online computational complexity of the User's algorithm in the protocol of Section 7.2 does not depend on the number of attributes.

*Diagnostic programs* evaluate the data dump produced by the runtime environment and diagnose the problem. In this chapter, we use diagnostic programs generated by the Clarify system [60]. We emphasize that Clarify is a "black-box" diagnostic system, and thus not simply an alternative to debugging. Commercial applications are often distributed as packed binaries, compiled without symbols and not accompanied by source code. Investigation of a fault in such a binary by manual debugging is a laborious process, whereas even an unsophisticated user can benefit from fast diagnostics provided by systems like Clarify.

In general, the diagnostic program can be manually created by human experts, or constructed automatically by training a machine learning classifier on previously labeled program behaviors (supplied, for example, by beta testers who are not concerned about privacy of their data). Clarify takes the latter approach. If necessary, standard methods for privacy-preserving decision-tree construction can be used to protect data suppliers' privacy [6,78]. The number of users whose data are used in constructing the diagnostic program is typically

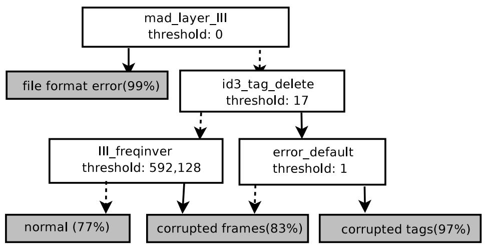


Figure 7.1: Diagnostic branching program for the `mpg321` benchmark. Dotted lines are taken when the attribute (normalized count of the feature value) is less than or equal to the threshold listed in the box, while the solid line is taken when it is greater than the threshold. The threshold is determined automatically for each benchmark by the decision-tree algorithm, and can be different for each node in the tree. Clear boxes are decision nodes. Shaded boxes are classification nodes.

orders of magnitude lower than the number of users who apply the resulting program to their data. Therefore, we focus on privacy-preserving *evaluation* of diagnostic programs.

The diagnostic program usually has the form of a classification tree or a branching program. In each internal node, one of the attributes is compared with some threshold value. Leaves contain diagnostic labels. For example, Figure 7.1 shows a diagnostic branching program created by Clarify using function counting for the mp3 player `mpg321`. The application itself does not give any consistent error messages for any of these error cases. The model has four diagnostic labels, including normal execution (no error), file format error (trying to play a wav file as if it were an mp3), corrupted tag (mp3 metadata, *e.g.*, artist name is stored in ID3 format tags), and corrupted mp3 frame data.

The diagnostic branching program distinguishes between these three failure modes and normal execution.

At the root, the function `mad_layer_III` provides almost perfect discriminative information for the wav error class (trying to play a wav as if it were an mp3): the `mad_layer_III` routine is part of the `libmad` library and is called when the audio frame decoder runs. Since the *wav* format is among the formats not supported by `mpg321`, it will not successfully decode any audio frames, and the `libmad` library will never call `mad_layer_III`.

The `id3_tag_delete` routine differentiates between the corrupted tag and and other classes. The ID3 tag parser in the `libid3tag` library dynamically allocates memory to represent tags and frees them with `id3_tag_delete`. If tag parsing fails, the memory for a tag is not allocated. Since no tag parsing succeeds in the corrupted frames case, `id3_tag_delete` is never called to free the tag memory, making its absence discriminative for that class. The `libmad` audio library's default error handler `error_default` is used if the application does not specify one. `mpg321` does not specify its own error handler, so the presence of the function indicates corrupted audio frames, and its absence indicates corrupted id3 tags. Finally, `III_freqinver`, which performs subband frequency inversion for odd sample lines, is called very frequently as part of normal decoding of audio frame data. When there are corrupted frames, this function is called less frequently, and the decision-tree algorithm finds an appropriate threshold value to separate the normal from the corrupted case.

| App. | Attributes | Nodes | # Errs | Accuracy |
|---|---|---|---|---|
| gcc | 2,920 | 37 | 5 | 89.2% |
| latex | 395 | 1,107 | 81 | 85.1% |
| mpg321 | 128 | 9 | 4 | 87.5% |
| nfs | 292 | 17 | 5 | 93.3% |
| iptables | 70 | 9 | 5 | 98.5% |

Table 7.2: For each benchmark, we give the length of the attribute vector, number of nodes in the diagnostic tree, number of errors distinguished by the decision tree, and the tree's classification accuracy.

Table 7.2 shows the parameters of diagnostic programs for several benchmark applications.

## 7.4 Performance

We evaluated our prototype implementation using PCs with an Intel Pentium D 3 GHz processor, 2 GB of RAM, and 2MB cache. This is a realistic approximation of what the User might use, but we expect that the Server would maintain a more powerful dedicated server to process remote diagnostics requests.

In our analysis of scaling behavior, we created artificial data sets with varying numbers of nodes and attributes, and measured the offline and online time separately. Offline time includes all calculations that can be performed independently of the other party, while online time includes the calculations that depend on the information sent by the other party earlier in the protocol. Online time is the more important metric, since it dictates how long the two parties must maintain a connection. Offline calculations can be performed

|  | Server |  | User |  |
|---|---|---|---|---|
| Application | Time | Bytes | Time | Bytes |
| `foxpro` (7 nodes, 224 attrs) | 1s | 119 KB | 2s | 78 KB |
| `iptables` (9 nodes, 70 attrs) | 2s | 155 KB | 2s | 61 KB |
| `mpg321` (9 nodes, 128 attrs) | 2s | 155 KB | 2s | 71 KB |
| `nfs` (17 nodes, 292 attrs) | 2s | 298 KB | 4s | 142 KB |
| `gzprintf` (23 nodes, 60 attrs) | 3s | 506 KB | 5s | 133 KB |
| `gcc` (37 nodes, 2920 attrs) | 5s | 656 KB | 7s | 707 KB |
| `latex` (1107 nodes, 395 attrs) | 113s | 19,793 KB | 189s | 5,908 KB |

Table 7.3: Privacy-preserving evaluation of Clarify diagnostic programs: computation and communication costs.

during idle times when the CPU is in low demand.

We first analyze the scaling behavior of the Server algorithm, as presented in Figures 7.2 and 7.3. Here we see that the Server's computation and bandwidth requirements are independent of the number of attributes, which is an attractive property in the software diagnostic scenario where the attribute vector can be quite large and contain a lot of information which is not relevant for all diagnostic programs. Furthermore, the Server's algorithm scales linearly with the number of nodes in the branching program, which is as good as one can realistically hope to achieve.

Scaling behavior of the User's algorithm is shown in Figures 7.4 and 7.5.

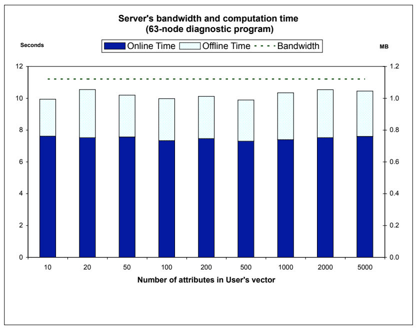


Figure 7.2: Server algorithm: scaling with the number of attributes.

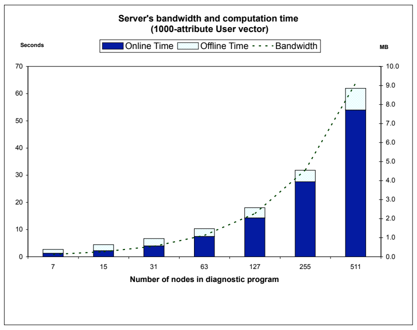


Figure 7.3: Server algorithm: scaling with the size of the diagnostic program.

As is the case with the Server, the User's online computation time depends linearly on the size of the branching program, but is independent of the number of attributes in the attribute vector. Unlike the Server, the User's offline computation time and bandwidth requirements do depend on the number of attributes. This is because the User must encrypt the entire attribute vector offline, and then transmit it as part of the protocol.

We also evaluate our prototype implementation on several real applications, as shown in Table 7.3. These benchmarks have been chosen because they are common, heavily-used programs that either contain security vulnerabilities which would be revealed by the diagnostic program (*e.g.*, `gzprintf`), or report misleading error messages (or none at all) for non-exotic error conditions, and therefore would benefit the most from remote diagnosis. As our diagnostic programs, we used classification trees generated by Clarify [32, 60], and as attribute vectors 32-bit invocation counters for each function of the

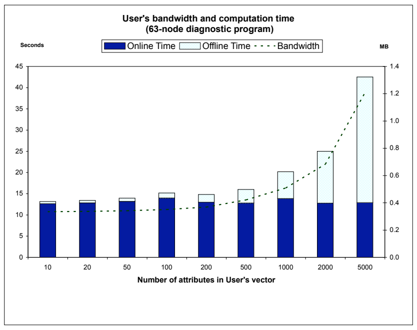


Figure 7.4: User algorithm: scaling with the number of attributes.

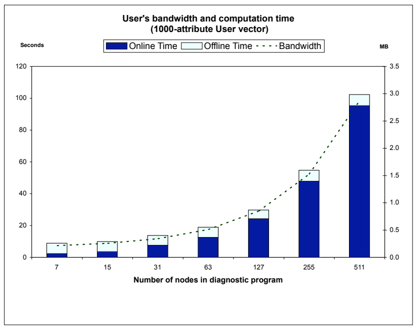


Figure 7.5: User algorithm: scaling with the size of the diagnostic program.

application.

For all applications, the computation and communication cost of executing our privacy-preserving protocol is acceptable in many practical scenarios.

## 7.5 Conclusions

We presented a practical, provably secure protocol which enables a User to evaluate the Server's branching program on the User's local data without revealing any information except the diagnostic label. We applied our prototype implementation to several realistic benchmarks, using diagnostic decision trees produced by the Clarify system as our branching programs, and demonstrated that it performs well in many practical scenarios.

# Chapter 8

# Privacy-Preserving Decision-Tree Learning

## 8.1 Introduction

Privacy-preserving data analysis is one of the most important applications of secure multi-party computation. In this chapter, we develop a privacy-preserving version of a fundamental data-analysis primitive: an algorithm for constructing or learning a *classifier*. Classifiers, such as decision trees, are a mainstay of data mining and decision support [103]. Given a database with multiple attributes (an attribute can be thought of as a column in a database schema), a classifier *predicts* the value of a "target" or "class" attribute from the values of "feature" attributes. One can also think of a classifier as assigning records to certain classes (defined by the value of the class attribute) on the basis of their feature attributes. A popular machine-learning task is to automatically learn a classifier given a training set of records labelled with class attributes. Classifiers built in this way are used for marketing and customer relationship management, development of better recommendation algorithms and services, clinical studies, and many other applications.

We focus on the problem of securely constructing a classifier in a two-party setting where one party provides a database, while the other party provides the parameters of the classifier that it wants to construct from the records in the database. This is a common situation in law-enforcement, regulatory, and national-security settings, where the entity performing the analysis (for example, an agency investigating irregular financial transactions) does not want to reveal which patterns it is mining the database for (for example, to prevent the target of investigation from structuring their transactions so as to avoid scrutiny). Confidentiality of the resulting classifier is also important in scenarios where both the data-analysis techniques and the output of the analysis process constitute potentially valuable intellectual property, *e.g.*, when mining patient databases in clinical studies, constructing expert systems and diagnostic frameworks, and so on.

The key privacy properties that the protocol for privacy-preserving classifier learning must guarantee are, informally, as follows. First, the records from which the classifier is constructed should remain confidential from the party who obtains the classifier (except for the information which is inevitably revealed by the classifier tree itself). Second, the data owner should not learn anything about the classifier which has been constructed. While the algorithm for constructing the classifier is standard (*e.g.*, ID3), its parameters—(i) which attributes are used as features?, (ii) which attributes are used as class attributes?, (iii) if the classifier is being constructed only on a subset of database records, what is the record selection criterion?—should remain hidden from the data owner. Note that the latter requirement precludes the data owner from simply computing the classifier on his own.

Previous work on privacy-preserving classifier learning [78, 116, 117] focused on a very different problem in which the resulting classifier is revealed to *both* parties. This greatly simplifies the protocol because the classifier can be constructed using the standard recursive algorithm—since both parties learn the resulting classification tree, revealing each node of the tree to both parties as it is being constructed does not violate the privacy property. This is no longer true in our setting, which presents a non-trivial technical challenge.

Existing protocols cannot be used in practical scenarios where confidentiality of the classifier is essential. For example, a national-security agency may want to mine records of financial transactions without revealing the classified patterns that it is looking for (defined by its choice of feature and class attributes and of a certain subset of individuals in the database). Other scenarios include construction of a recommendation algorithm from transactional data without revealing it prematurely (*e.g.*, the Netflix Prize competition [98]); clinical studies involving competing medical institutions, each of which is fiercely protective both of their patient data *and* their analysis techniques (which subset of patients to look at, which symptoms to focus on, and so on), because the latter can lead to patentable and potentially lucrative diagnostic methods; expert systems, where the classifier constitutes valuable intellectual property; remote software fault diagnostics; and many others.

In this chapter, we use the same basic framework of secure multi-party computation (SMC) as the original paper on privacy-preserving data mining by Lindell and Pinkas [78] and aim to provide the same level of cryptographic security guarantees. We emphasize, however, that (i) our desired privacy properties (in particular, confidentiality of the resulting classifier) are very different and more challenging because the techniques of [78] no longer work; (ii) we allow, but do not assume or require that the data are partitioned between the two parties; and (iii) unlike [78], we provide a prototype implementation and performance measurements in order to evaluate the scalability of the SMC-based approach to privacy-preserving data classification.

**Our contributions.** We present a cryptographically secure protocol for privacy-preserving construction of classification trees. The protocol takes place between a user and a server. The user's input consists of the parameters of the classifier that he wishes to construct: which data attributes (columns) to use as feature attributes, which as the class attribute, and, optionally, which predicate on records (rows) to use in order to select only a subset of the database records for the classifier construction. The server's input is a relational database. We assume that the schema of the database (*i.e.*, names of attributes and the values they can take) is public, but that the actual records are private.

The user's protocol output is a classification tree constructed from the server's data. The server learns nothing from the protocol; in particular, he does not learn the parameters of the classification algorithm, not even which attributes have been used when constructing the classifier. We re-iterate that the latter requirement precludes the server from computing the classifier on his own, and also makes existing protocols inapplicable.

Our protocol exploits the structure of the classifier-construction algorithm in a fundamental way. In each node of the classification tree, the records are "split" based on the value of some attribute. In order to pick the best attribute for this purpose, the tree-construction algorithm must, in each node of the tree, count the number of records that fall into several categories. In contrast to [78], the database owner should not learn how many of his own records fall into each category, so we must perform this computation in a privacy-preserving manner. If done naïvely, using generic techniques, the computational cost of the resulting protocol would be prohibitive.

Our key technical innovation is to build the tree "one tier at a time" by simultaneously counting the categories for an entire tier of nodes rather than for a single node. By partitioning the categories into mutually exclusive groups, we are able to compute the counts for a whole tier of nodes using the same number of secure circuit evaluations as we would have needed for a single node. This enables a substantial performance gain which bridges the gap between theoretical and practical efficiency.

Our final contribution is to measure the scalability of our prototype implementation and evaluate its performance on realistic datasets. While theoretical protocol designs in the SMC framework abound, actual implementations have been very rare. This makes it difficult to determine whether these (theoretically sound) techniques can actually be applied, even given modern computing power, to anything other than toy examples. Our performance measurements show the limits of SMC-based privacy-preserving data analysis.

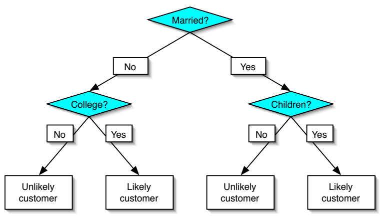


Figure 8.1: Example decision tree.

## 8.2 Problem Formulation

### 8.2.1 Decision-tree learning

A *classifier* takes as input a *record* (or transaction) consisting of several attribute values, and outputs a classification label which categorizes the record. *Decision trees* are a common type of classifiers. Each internal node in a decision tree considers a single attribute and redirects evaluation to one of several child nodes based on the value of that attribute. Once a leaf node is reached, the classification label contained therein is output as the result of classification. Fig. 8.1 shows an example decision tree that could be used by a marketing department to determine whether a consumer is likely to buy a company's product.

Decision-tree classifiers can be constructed manually by a human expert with domain knowledge, but algorithms for *decision-tree learning* are increasingly popular (*e.g.*, see Algorithm 6). Given a database of records tagged with classification labels, the algorithm constructs the decision tree recursively from the top down. At the root node, the algorithm considers every attribute and measures the quality of the split that this attribute will provide (see below).

```
Input:
$\mathcal{R}$, the set of feature attributes.
$C$, the class attribute.
$T$, the set of records.
$d$, the current depth.
$D$, the desired maximum depth.
DecisionTree($\mathcal{R}, C, T, d, D$)
1: if $d = D$ or $\mathcal{R}$ is empty then
2:     return a leaf node with the most frequent class label among the records
       in $T$.
3: else
4:     Determine the attribute that best classifies the records in $T$, let it be $A$.
5:     Let $a_1, ..., a_m$ be the values of attribute $A$ and let $T(a_1), ..., T(a_m)$ be a
       partition of $T$ such that every record in $T(a_i)$ has the attribute value $a_i$.
6:     Return a tree whose root is labeled $A$ (this is the splitting attribute)
       and which has edges labeled $a_1, ..., a_m$ such that for every $i$, the edge $a_i$
       goes to the tree DecisionTree($\mathcal{R} - \{A\}, C, T(a_i), d + 1, D$).
7: end if
```

**Algorithm 6:** The (non-private) recursive decision-tree learning algorithm.

The algorithm chooses the "best" attribute and partitions all records by the value of this attribute, creating a child node for each partition. The algorithm is then executed recursively on each partition.

Two popular measures of the "quality" of a split are information gain and the Gini index. Information gain is used in the ID3 and C4.5 algorithms [103], while the Gini index is used in the CART algorithm [16]. Information gain can be computed privately using the $x \log x$ protocol from [78]. Our privacy-preserving protocol for decision-tree learning can use either, but the private computation of the Gini index is more efficient, so we will focus on it.

In the following, suppose that the class attribute (*i.e.*, the target of classification) can assume $k$ different values $c_1, ..., c_k$ and that the candidate splitting attribute $A$ can assume $m$ different values $a_1, ..., a_m$. Denote by $p(c_i)$ the portion of the records whose attribute $C = c_i$, by $p(a_i)$ the portion of the records whose attribute $A = a_i$, and by $p(c_i|a_j)$ the portion of the records that have both attribute $C = c_i$ and attribute $A = a_i$.

The Gini index $\textsc{Gini}(A)$ is computed as:

$$1 - \sum_{i=1}^{k} (p(c_i))^2 - \sum_{j=1}^{m} p(a_j) \sum_{i=1}^{k} p(c_i|a_j)\,(1 - p(c_i|a_j)) \tag{8.1}$$

If we use the notation $n(c_i)$ for the *number* of records with attribute $C = c_i$, then we can rewrite (8.1) as:

$$1 - \sum_{i=1}^{k} \left(\frac{n(c_i)}{|T|}\right)^2 - \sum_{j=1}^{m} \frac{n(a_j)}{|T|} \sum_{i=1}^{k} \frac{n(c_i|a_j)}{|T|} \left(1 - \frac{n(c_i|a_j)}{|T|}\right) \tag{8.2}$$

Multiplying this equation by $|T|^3$ gives:

$$|T|^3 - \sum_{i=1}^{k} (n(c_i))^2\,|T| - \sum_{j=1}^{m} n(a_j) \sum_{i=1}^{k} n(c_i|a_j)\,(|T| - (c_i|a_j)) \tag{8.3}$$

Since the number of records $|T|$ is fixed, we can compare the Gini index of different attributes using only multiplication and addition. These operations can be easily computed in a privacy-preserving manner using Yao's garbled-circuits method.

### 8.2.2 Distributed decision-tree learning

Conventional decision-tree learning is performed by a single user. The user has access to some database $T$ and chooses the set of feature attributes

$\mathcal{R}$, the class attribute $C$, and the number of tiers $D$. In this chapter, we focus on a *distributed* setting, where the database $T$ resides on a server and a remote user chooses $\mathcal{R}$, $C$, and $D$. We emphasize that for real-world databases, where the total number of attributes is fairly large, $\mathcal{R}$ may be only a small subset of attributes. For example, attributes of $T$ may include hundreds of demographic features, and the user may be interested only in a handful of them for classification purposes.

In the distributed setting, both parties may have privacy concerns. The server wishes to reveal no more about $T$ than is necessarily revealed by a decision tree based on $T$. The user, on the other hand, may not wish to reveal which feature attributes $\mathcal{R}$ and class attribute $C$ he selected for the purposes of constructing a classifier.

We assume that several parameters are known to both parties: $|T|$, the number of records in the database; $\mathcal{A}$, the set of all attributes in the database; the set $a_1, ..., a_m$ of possible values for each attribute $A \in \mathcal{A}$; $|\mathcal{R}|$, the number of feature attributes selected by the user; and $D$, the depth of the decision tree to be constructed.

**Branching factor.** In the general case, the record database $T$ may contain nominal attributes whose domains have different sizes. For instance, a consumer database may have 2 possible values for the "sex" attribute, and 50 possible values for the "state of residence" attribute. We refer to the number of different values that an attribute can assume as its *branching factor*, because it determines the number of children for each internal node corresponding to that attribute.

When the decision tree is computed in a privacy-preserving manner, all internal nodes must have the same number of children in order to prevent the server from learning which attribute is considered in a given node. Therefore, all attributes must have the same branching factor $m$. As a pre-processing step, attributes can be padded with unused values so that all attributes have the same branching factor. For simplicity, we assume that each attribute value is encoded as an integer between 0 and $m-1$, and can thus be represented using $\log_2 m$ bits.

## 8.3 Privacy-Preserving Evaluation of Decision Trees

Our protocol for privacy-preserving decision-tree *learning* requires that we have a protocol to privately *evaluate* decision trees. We use the protocol from Chapter 7, with several substantial modifications. In the protocol from Chapter 7, attributes can take one of a large number of different values, and each internal node selects one of two children based on a threshold comparison. In this chapter's setting, each attribute takes one of $m$ values ($m$ is relatively small), and internal nodes have $m$ children—one for each attribute value.

The privacy requirement is that this evaluation should be oblivious: the evaluator should not learn anything about the structure of the tree except the total number of nodes and the length of the evaluation path, nor *which* of his attributes were considered during evaluation. To achieve the former, the tree is represented as a set of encrypted nodes; decrypting each node reveals the index of the next node (which depends on the value of the attribute considered in the parent node) and the corresponding decryption key. To hide which attribute is considered in each node, the "oblivious attribute selection" protocol from the previous chapter splits each of the attributes that will be used during evaluation into two random shares. The circuit creator receives one share and the evaluator receives the other, without learning to which of his attributes this share corresponds.

Each oblivious evaluation of an internal node results in moving control to one of the $m$ child nodes. While in the previous chapter we used 1-out-of-2 conditional oblivious transfer, now we must use 1-out-of-$m$ conditional oblivious transfer. For details, see Section 2.5.1.

We need a technical trick so that the decision-tree evaluation protocol can be efficiently invoked multiple times on the same set of attributes. Recall that as the result of oblivious attribute selection, the evaluator has a random share for each of his attributes that will be used in some internal decision node. We use the circuit logic shown in Algorithm 7 for internal nodes. Since the evaluator provides as input his shares for *all* attribute values, he cannot tell which one was actually selected and combined with the share from the circuit creator to obtain the complete attribute value.

Modifying the circuit logic in this way ensures that the evaluator's input is the same for all nodes of all trees created during our protocol. This enables a substantial efficiency gain. Instead of generating random wire keys

```
**Creator's Input:**
$i$, $1 \leq i \leq r$, the attribute index
$a_i^C$, the Creator's share of attribute value $a_i$
**Evaluator's input:**
$a_1^E, ..., a_r^E$, the Evaluator's shares for all attribute values
**Output:**
The value $a_i$, using $\log_2 m$ wires.
InternalNodeGate($i, a_i^C, a_1^E, ..., a_r^E$)
 1: return $a_i^E + a_i^C \pmod{m}$
```
**Algorithm 7:** Modified circuit logic for internal nodes

for each bit of the evaluator's input into each circuit (as in the standard Yao's method), we generate them once, and then re-use this representation for the evaluator's input wires in all circuits. This allows us to perform only a single set of oblivious transfers to provide the evaluator with the the wire keys corresponding to his input bits. These wire keys are then used in all of the garbled circuits.

## 8.4 Privacy-Preserving Decision-Tree Learning

Our protocol takes place between a server in possession of a database $T$ and a user who wishes to build a classifier for class attribute $C$ based on a set $\mathcal{R}$ of feature attributes. The tree is constructed from the root down, as in the conventional algorithm shown in Fig. 6. Unlike the conventional algorithm, however, ours is non-recursive. Instead, the tree is constructed one tier at a time. When processing tier $i$, $m^i$ pending nodes are considered. In the final tier, the pending nodes are transformed into leaf nodes with classification labels in them; in all intermediate tiers, they become internal decision nodes, where the attribute for making the decision is chosen based on the data in $T$. We now describe the protocol, which is divided into four phases.

### 8.4.1 Phase 1: Sharing the attribute values

The set of attributes $\mathcal{A}$ found in the database $T$ may be far larger than the set $\mathcal{R} \cup \{C\}$ of attributes that are relevant to tree construction. For an attribute $R_i \in \mathcal{R} \cup \{C\}$ and a record $t \in T$, $t[R_i]$ refers to the attribute value for attribute $R_i$ in record $t$. For each record $t \in T$ and for each relevant attribute $R_i \in \mathcal{R} \cup \{C\}$, Phase 1 enables the user and the server to learn shares $t[R_i]_U$ and $t[R_i]_S$ such that $t[R_i]_U + t[R_i]_S \pmod{m} = t[R_i]$. This is done using the oblivious attribute selection technique from [17], which is outlined below:

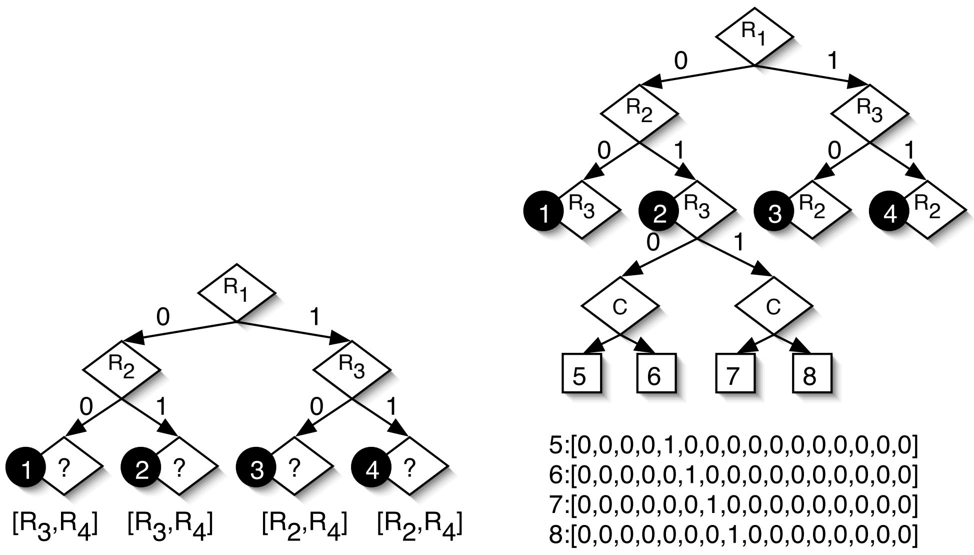
1. For all $A_i \in \mathcal{A}$, the server encrypts $t[A_i]$ using an additively homomorphic encryption scheme, and sends $\{t[A_i]\}_y$ to the user.
2. User creates a blinding value $b_i$ for each relevant attribute $R_i$, and uses the homomorphic property to add $b_i$ and $t[R_i]$ under encryption. User sends $\{b_i + t[R_i]\}_y$ to the server. User's random share is $t[R_i]_U = -b_i \bmod m$.
3. Server decrypts to obtain $b_i + t[R_i]$ and stores $t[R_i]_S = b_i + t[R_i] \mod m$.

We use a blinding value $b_i$ at least 80 bits longer than the $(\log_2 m)$-bit value $t[R_i]$ so that it statistically hides $t[R_i]$. The shares $t[R_i]_U$ and $t[R_i]_S$ will be used in later phases as inputs to small Yao circuits that are generated by the user and evaluated by the server. Therefore, the server needs to learn the random wire keys representing his input shares $t[R_i]_S$ in the circuit. As usual, this is done via a 1-out-of-2 oblivious transfer for each of the $\log_2 m$ bits in the $t[R_i]_S$ values, where the server's input is the $j$th bit of $t[R_i]_S$, and the user's input is the pair of wire keys representing, respectively, 0 and 1 on the input wire corresponding to this bit.

Unlike the standard Yao protocol, the same input-wire keys are used for multiple circuits. The oblivious transfers can thus be done only once per protocol execution instead of once per circuit evaluation. This results in a substantial performance improvement, since the bulk of computation in secure circuit evaluation is spent on the oblivious transfers.

We also observe that our protocol can be applied not only in the case where the server holds the entire database, but also for any vertical or horizontal partitioning of the database between the user and the server. If the database is partitioned, the steps described above are carried out only for the attribute values held by the server. For each value held by the user, the user simply splits it into two random shares and sends one of them to the server. Regardless of the database partitioning, after Phase 1 every attribute value of every record is shared between the user and the server.

After performing these preliminary steps, the user participates in the PrivateDecisionTree$(\mathcal{R}, C, D, T)$ protocol with the server, which starts Phase 2.



(a) Incomplete Tree (b) Augmented Tree

Figure 8.2: An incomplete decision tree with 4 pending nodes, and the same tree augmented with a feature attribute and class attribute

### 8.4.2 Phase 2: Computing category counts

Phase 2 is shown in Algorithm 8 as lines 3–6.

Let $d$ be the depth of the current tier. Within this tier there are $m^d$ pending nodes, and of the original $|\mathcal{R}|$ feature attributes, only $|\mathcal{R}| - d$ remain as candidates to be chosen as the splitting attribute for each pending node because $d$ attributes have already been used. The set of candidate attributes for splitting at a pending node $n$ depends on which attributes were already encountered on the path from the root node to $n$, and may thus be different for each pending node. For example, Fig. 8.2(a) shows a tree entering Phase 2 on tier 2. The path to the 3rd pending node consists of the edges $R_1 = 1$

and $R_3 = 0$, so attributes $R_1$ and $R_3$ are no longer available as candidates for this node. The candidates for the 3rd pending node are $[R_2, R_4]$, while the candidates for the 2nd pending node are $[R_3, R_4]$,

Let $T(n)$ be the set of records that satisfy the preconditions of node $n$ (for the 3rd pending node in Fig. 8.2(a), these are records with $R_1 = 1$ and $R_3 = 0$). Let $\{R_{n_1}, ..., R_{n_{|\mathcal{R}|-d}}\}$ be the set of candidate attributes for node $n$. Finally, let $T_k(n : i, j)$ be the set of records in $T(n)$ that have $R_{n_k} = i$ and $C = j$. To determine the quality of the split that would be provided by choosing $R_{n_k}$ as the splitting attribute for this node, it is necessary to compute $|T_k(n : i, j)|$ for all possible values of $i$ and $j$ ($0 \le i, j \le m$).

For any choice of $n$, $i$, and $j$, the user can build a decision tree to determine whether a given record is in $T_k(n : i, j)$. Using oblivious decision-tree evaluation, the user and the server can then learn shares of $|T_k(n : i, j)|$ without either revealing his private inputs. The problem with this naïve approach is that determining the quality of splitting on a single attribute $R_{n_k}$ requires $m^d \cdot m^2$ oblivious decision-tree evaluations on each record in $T$ (one for each choice of $n$, $i$, and $j$).

Our construction is significantly more efficient because it iterates over the database only *once* by counting $m^d \cdot m^2$ different mutually exclusive categories simultaneously. The key observation is that for each record $t \in T$, there is a unique pending node $n$ such that $t \in T(n)$. Furthermore, for each $t \in T(n)$ and $0 \le k \le |\mathcal{R}| - d$, there are unique $i, j$ such that $t \in T_k(n : i, j)$.

Therefore, our construction builds a classifier to determine for which values of $n$, $i$, and $j$ the record $t$ belongs to $T_k(n : i, j)$.

To do this, we augment the partially constructed tree $P$ by replacing each pending node with a depth-two subtree that considers attributes $C$ and $R_{n_k}$. Fig. 8.2(b) shows the result of augmenting the tree from Fig. 8.2(a) when $k = 1$. (To avoid clutter, the augmented portion is only shown for the 2nd pending node.) The $m^d \cdot m^2$ leaves of the tree contain vectors of length $m^d \cdot m^2$ as their labels. Each leaf is reachable by records in $T_k(n : i, j)$ for a unique choice of $n$, $i$, and $j$, and the vector used as its label has a single "1" in the position corresponding to $T_k(n : i, j)$ and "0" elsewhere.

Once the augmented tree $P' = P.\textsc{AugmentWithAttAndClass}(k, C)$ has been constructed, the user and server engage in a privacy-preserving decision-tree evaluation protocol for each record $t \in T$. To support oblivious evaluation, the tree must be transformed as follows (see Chapter 7 and Section 8.3 for details). Each node other than the root is encrypted with a random key. Each internal node is replaced by a small Yao circuit that takes as its input the user's and server's shares $t[R_i]_U$ and $t[R_i]_S$ of the relevant attribute values $t[R_i]$ for each $R \in \mathcal{R}$, and outputs the index and decryption key for the appropriate child node. Each leaf node has as its label a vector of $m^d \cdot m^2$ values, encrypted using a user-created instance of an additively homomorphic encryption scheme. As described above, the vector has "1" in the position corresponding to its category, and "0" in all other positions.

Note that although the same tree is applied to every record, it must be freshly transformed into a secure tree for each oblivious evaluation.

As the result of oblivious evaluation of augmented trees, the server learns a vector of $m^d \cdot m^2$ ciphertexts. All but one are encryptions of "0." The sole ciphertext encrypting "1" occurs in the position corresponding to the category of the record (of course, the server cannot tell which ciphertext this is). By summing up these vectors under encryption, the server obtains ciphertexts encrypting the counts $|T_k(n : i, j)|$. The server must then transform these encrypted counts into additive random shares (mod $|T|$), using the same technique as in Sect. 8.4.1.

The following subroutines are used during Phase 2:

- $P$.AugmentWithAttAndClass. This method is executed by the user, and adds two tiers to the tree $P$: one for the attribute $R_{n_k}$ (different for each pending node) and one for the class attribute $C$.
- Prot:EncryptedCounts. This protocol between the user and the server results in the user and server holding shares for the counts $|T_k(n : i, j)|$ for $n$=1 to $m^d$, $i$=1 to $m$, and $j$=1 to $m$. Pseudocode is given in Algorithm 9.

### 8.4.3 Phase 3: Selecting the highest-quality split

Phase 3 is shown in Algorithm 8 as lines 7–13.

**User's Input:**
$\mathcal{R}$, the set of feature attributes ($|\mathcal{R} > D$)
$C$, the class attribute
$D$, the desired maximum depth
**Server's Input:** $T$, the set of records converted into random wire values.
**User's Output:** $P$, a decision tree to classify $C$ from $\mathcal{R}$
$\textsc{Prot:PrivateDecisionTree}$(user: $\mathcal{R}, C, D$ server: $T$)

```
1:  $P$ = new tree
2:  for $d$=0 to $D-1$ do
3:      for $k$=1 to $|\mathcal{R}|-d$ do
4:          $P'$=$P.\textsc{AugmentWithAttAndClass}(k, C)$
5:          $(|T_k(...)|_U, |T_k(...)|_S)$ = $\textsc{Prot:EncryptedCounts}$(user: $P'$ server: $T$)
6:      end for
7:      for $n$=1 to $m^d$ do
8:          for $k$=1 to $|\mathcal{R}|-d$ do
9:              $(Q_U^k, Q_S^k) = \textsc{Prot:ComputeQuality}(|T_k(n:...)|_U, |T_k(n:...)|_S)$
10:         end for
11:         bestatt = $\textsc{Prot:ArgMax}$ (user: $Q_U^1, ..., Q_U^{|\mathcal{R}|-d}$ server: $Q_S^1, ..., Q_S^{|\mathcal{R}|-d}$)
12:         In $P$, make node $n$ an internal node splitting on attribute $R_{n_{\text{bestatt}}}$
13:     end for
14: end for
15: $P' = P.\textsc{AugmentWithClass}(C)$
16: $(|T(...)|_U, |T(...)|_S) = \textsc{Prot:EncryptedCounts}$(user: $P'$ server: $T$)
17: for $n$=1 to $m^D$ do
18:     bestclass = $\textsc{Prot:ArgMax}$(user: $|T(n:*,1)|_U, ..., |T(n:*,m)|_U$
                                      server: $|T(n:*,1)|_S, ..., |T(n:*,m)|_S$)
19:     In $P$, make node $n$ a leaf node with label bestclass
20: end for
21: return p
```

**Algorithm 8:** The private "one-tier-at-a-time" decision-tree learning protocol

**User's Input:** A decision tree $P$ with $k$ leaf-nodes. The label of leaf $i$ is a $k$-length vector with $\{1\}_y$ in position $i$ and $\{0\}_y$ in all other positions.
**Server's Input:** A record set $T$ for which each bit of each attribute value has been converted into a random wire value.
**Output:** Let $K = \sum_{t \in T} P(t)$ be the $k$-length vector whose $i$th entry is the number of records in $T$ landing in leaf node $i$. The user's and server's outputs are shares $K_U$ and $K_S$ of $K$.

```
EncryptedCounts($P, T$)
1: $K \leftarrow$ length $k$ vector with each entry set to $\{0\}_y$
2: for each $t \in T$ do
3:     $J \leftarrow$ PrivateTreeEval($P, t$)
4:     $K \leftarrow K + J$ under encryption
5: end for
6: Split each component of $K$ into shares; user decrypts his share
```

**Algorithm 9:** Protocol to determine how many records fall into each of $k$ categories

After Phase 2, the user and the server share counts $|T_k(n : i, j)|$ for all pending nodes $n$ in the tier, and for all values of $k$, $i$, and $j$. This enables them to compute $\mathrm{G}ini(R_{n_k})$ for each node $n$ using (8.3), but over $T(n)$ rather then the entire record set $T$. The user and server must compute

$$|T(n)|^3 - \sum_{j=1}^{m} |T(n)|\,|T_k(n : *, j)|^2 - \\ \sum_{i=1}^{m} |T_k(n : i, *)| \sum_{j=1}^{m} |T_k(n : i, j)|\,(|T(n)| - |T_k(n : i, j)|) \quad .$$

In the above, $|T(n : *, j)|$ is the number of nodes in $T(n)$ with class attribute $C = j$, and $|T_k(n : i, *)|$ is the number of nodes in $T(n)$ with attribute $R_{n_k} = i$. These values, along with $|T(n)|$, can be computed from the shares $|T_k(n : i, *)|_{\{U,S\}}$ which the user and server hold.

Given the shares of all inputs, a simple circuit produces shares of $Gini(R_{n_k})$ for each node $n$ and for each $k$, using only addition and multiplication. For each pending node $n$, these shares are then fed into another garbled circuit. This circuit determines which attribute $R_{n_k}$ provides the best split quality. The user updates the tree $P$ with this information, by replacing the pending node $n$ with an internal node that splits on the attribute $R_{n_k}$.

The following subroutines are used during Phase 3:

- Prot:ComputeQuality. This protocol uses a garbled circuit to compute the Gini index for node $n$ and attribute $R_{n_k}$. This protocol takes as input shares of the $m^2$ counts $|T_k(n : i, j)|$, and returns shares of the Gini index.
- Prot:ArgMax. This protocol takes as input shares of values $v_1, ..., v_n$ and provides the user with an index $m$ such that $v_m$ is greater than or equal to all other values. The server learns nothing.

### 8.4.4 Phase 4: Constructing the bottom tier

Phase 4 is shown in Algorithm 8 as lines 17–20.

Phase 4 completes the decision tree $P$ by adding the correct labels to its leaf nodes. Each leaf node $n$ should have as its label the most common classification value among the records in $T(n)$. Similar to Phase 2, we can find the most popular classification value for all leaf nodes at once. The incomplete tree $P$ is augmented with a single extra tier which examines the classification node $C$. Then EncryptedCounts provides the user and server with the shares of the the counts $|T(n:*,j)|$ for $n$=1 to $m^D$ and $j$=1 to $m$. Next, a garbled circuit finds the value $c$ such that $|T(n:*,c)|$ is maximal, and makes it the label for node $n$.

The following subroutines are used during Phase 4:

- $P$.AugmentWithClass. This is executed by the user and adds one additional tier to the tree $P$ for the class attribute $C$.
- Prot:EncryptedCounts. Same as in Phase 2, and provides the user and server with shares of $|T(n:*,j)|$.
- Prot:ArgMax. Same as in Phase 3.

### 8.4.5 Horizontal selection

In many applications of decision-tree learning, the user wants to construct a classifier using the records defined by a certain predicate, *i.e.*, from a *horizontal* subset of the database. In other words, the user selects not only a subset of columns to use as features, but also a subset of records (rows), and the protocol should construct a classifier using the data in the selected records only.

This is motivated by real-world scenarios. For example, a proprietary database may contains records for diverse individuals living throughout a nation, while the user is interested in building a marketing classifier only for consumers from a particular region or those belonging to a particular demographic. In this scenario, the user may wish to keep his record selection criterion private so as to avoid revealing his marketing strategy to competitors. Previous protocols for privacy-preserving decision-tree learning cannot solve this problem because, by their design, they reveal the resulting classifier to all protocol participants.

In this scenario, we assume that the user does not have a vertical partition of the database and, since he does not have access to the database, cannot explicitly specify the indices of the records which satisfy his selection criterion. Instead, he must choose them *implicitly* by providing a selection predicate to be evaluated on all records in the database. Clearly, the user wants to keep this predicate private from the server, while the server wants to keep the attribute values of each record private from the user. Depending on the scenario, the number of records which satisfy the predicate may need to be revealed to the user, to the server, to both, or to neither.

We outline an extension to our protocol for only one of these four variants, in which the number of satisfying records is revealed to the user but not to the server. This variant has some useful properties: the user may not believe that the classifier is of high quality if it is based on too few records (thus it is helpful for the user to know how many records were used in constructing the tree), while the server learns a significant amount of information about the user's predicate if he learns the number of records which satisfy the predicate (thus the user may prefer to have this number hidden from the server). This particular variant does present some privacy risks to the server: if the predicate, which is hidden from the server, selects a very small subset of records, then the resulting decision tree will leak a lot of information about the records in the selected subset.

The extension involves two components: (1) an additional phase of the protocol, in which the user learns the indices of all records in the database that satisfy his selection predicate, and (2) a slight change to the category-counting phase to ensure that the records *not* selected by the user's predicate are not counted as belonging to any category, and thus do not participate in determining the best attributes for each internal decision node of the classifier.

To determine the indices of the records that satisfy the predicate, the user and the server engage in an instance of the oblivious decision-tree evaluation protocol described in Section 8.3. The user's predicate is represented as a decision tree which evaluates a record and labels it with true if it satisfies the predicate and false otherwise. This decision tree is then obliviously evaluated for each record in the database $T$. The protocol of Section 8.3 guarantees that the results are revealed only to the user, and not to the data owner.

The records *not* satisfying the predicate (*i.e.*, those which the user's predicate evaluated to false) should not be used when constructing the classifier. Recall from Sect. 8.4.2 that in order to determine the best splitting attribute for each internal node of the classifier, the user builds decision trees whose labels are vectors of ciphertexts that all encrypt "0," except for a single ciphertext—in the position corresponding to the record's category—that encrypts “1.” For the records that he wants to “turn off,” the user simply constructs the tree where the labels contain encryptions of “0” only. This effectively means that the corresponding record is not included in any of the $T_k(n : i, j)$ categories, and thus has no influence on the Gini index computation which is used to find the best splitting attribute.

## 8.5 Performance

Recall that there are $|\mathcal{A}|$ attributes, each of which has a branching factor of $m$; $|\mathcal{R}|$ feature attributes; $|T|$ transactions, and depth $D$. In evaluating the performance of our protocol, we distinguish between online and offline computations. Offline computations include generating $m^{d+2}$ homomorphic encryptions of “0” for each of the $|T|(|\mathcal{R}| - d)$ augmented decision trees used at tier $d$ (user); generating homomorphic encryptions of $|T||\mathcal{A}|$ attributes for oblivious attribute selection (server); garbling of circuits to compute the Gini index and ArgMax (server), and garbling of circuits to compute attribute selection (user). Note that the number of gates in these circuits depends on $|\mathcal{A}|$ and $T$ (Gini), $T$ and $m$ (ArgMax), and $|\mathcal{A}|$ and $m$ (attribute selection).

The following cryptographic operations must be performed online once per protocol execution: $|T||\mathcal{R}|$ homomorphic additions for oblivious attribute selection (user); $|T||\mathcal{R}|$ homomorphic decryptions (server); and $|T|(|\mathcal{R}| + 1) \log m$ 1-out-of-2 oblivious transfers so that the server can learn wire values for his attribute shares. In addition, the following are performed online to construct tier $d$ (with $m^d$ nodes): symmetric encryption of

$(\sum_{h=1}^{d+2} m^h)$ garbled nodes for each of $|T|(|\mathcal{R}| - d)$ augmented decision trees (user); $d + 2$ symmetric decryptions and evaluations of garbled attribute selection circuits for each of $|T|(|\mathcal{R}| - d)$ augmented decision trees (server); $|T|(m^{d+2})$ homomorphic additions (server); $m^{d+2}$ homomorphic decryptions (user); evaluation of $m^d(|\mathcal{R}| - d)$ garbled circuits to compute the Gini index at tier $d$ (user); and evaluation of $m^d$ garbled circuits for ArgMax at tier $d$ (user).

Because performance is often a concern when using secure multi-party computation techniques, we evaluated a prototype Java implementation of our protocol. Fig. 8.3 shows how the online time required by our protocol depends on several parameters of the decision-tree learning problem: the branching factor, the number of feature attributes, the number of tiers, and the number of records. Online time is *independent* of the number of attributes. This makes our protocol especially well-suited to scenarios where the set of feature attributes is a relatively small subset drawn from a very large set of total attributes.

To evaluate performance on real-world data, we applied our protocol to the "cars" dataset from the UC Irvine machine-learning repository. This dataset has 1728 records and 7 attributes with a branching factor of 4. We chose to build a tree with 5 feature attributes and 2 tiers. Table 8.1 shows the time consumed by different online components of our protocol.

This experiment demonstrates that, unlike generic techniques, our protocol can be successfully applied to problem instances of realistic size.

|  | sym. enc | sym. dec | homo. dec | homo. add | OTs | eval |
|---|---|---|---|---|---|---|
| user | 114s | 0s | 7.1s | 0.07s | 185.2s | 4.2s |
| server | 0s | 171s | 8.0s | 41.9s |  | 12.7s |

Table 8.1: Runtime for the "cars" dataset from the UC Irvine repository.

## 8.6 Conclusions

The field of privacy-preserving data mining has two approaches to the problem of executing machine-learning algorithms on private data. One approach sanitizes the data through suppression and generalization of identifying attributes and/or addition of noise to individual data entries. The sanitized version is then published so that interested parties can run any data-mining algorithm on it.

The other approach is to use cryptographically secure multi-party computation techniques to construct protocols that compute the same answer as would have been obtained in the non-private case. This approach has typically been applied when the relationship between the parties is symmetric: for example, the database is partitioned between them and the result of the protocol execution is that both parties learn the same output based on the joint database. By contrast, in the sanitization approach, the parties executing the data-mining algorithms do not have any data of their own, while the database owner obtains no output at all.

Even if the data-mining algorithms are the same (*e.g.*, classifier learning), the privacy-preserving versions for the two settings are substantially dif-

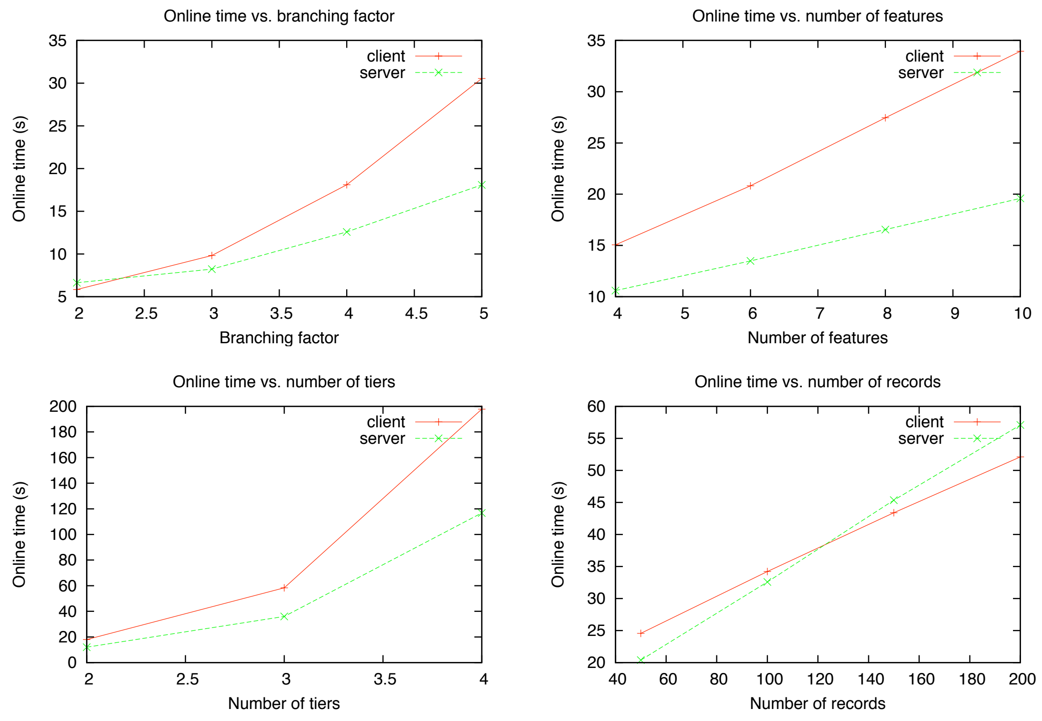


Figure 8.3: Online performance of the prototype implementation

ferent. We argue that settings where data are asymmetrically distributed and only one party learns the output are very natural in real-world scenarios. In this chapter, we show that it is possible to apply secure multi-party computation techniques to these scenarios. Our protocol requires several technical innovations (such as the ability to obliviously compute the sizes of several record categories in a single pass over the database). Unlike most designs in the literature, our protocol has been implemented, and we demonstrated that it can be efficiently applied even to problem instances of realistic size.

# Chapter 9

# Conclusions

In this thesis, we have examined several aspects of privacy-preserving data mining. In Chapter 4 we showed how a data miner can collect data from a group of individuals in such a way that the individuals are assured that the data miner does not know who provided which response. In Chapter 5 we examined the sanitized data publication paradigm, which alters data prior to release in order to preserve the privacy of constituent individuals. We showed how providing even modest privacy guarantees requires that the data be altered to the point that it is nearly useless.

By providing prototype implementations and performance statistics, this thesis shows that secure multi-party computation protocols are not only of theoretical interest, and can in fact be implemented and applied to real-world problems of modest size. Furthermore, this thesis bridges the gap between the scenarios that are typically considered in the sanitized publication and secure multi-party computation frameworks. The sanitized publication literature has focused on scenarios where the data is controlled by one party, and the algorithm that is to be run on the data is controlled by a different party. By contrast, the secure multi-party computation literature has focused on scenarios where the algorithm is known to all parties, and where the data is partitioned between parties. This has made direct comparisons of sanitized publication and secure multi-party computation approaches difficult, because the two approaches are used to solve similar problems but in completely different scenarios. In this thesis, Chapters 7 and 8 show how secure multi-party computation can be applied to problems where one party owns a secret algorithm, and another party owns secret data.

Privacy-preserving data mining remains a difficult problem. Real-world solutions will need to rely on a combination of legal, regulatory, and technological components. It is unlikely that we will ever reach a point where technological solutions alone can completely guarantee the privacy of individuals while allowing for meaningful exploratory data mining. Nevertheless, algorithms such as those developed in this thesis remain useful because they precisely identify what security guarantees are possible under different scenarios. This provides a framework that allows legal regulations to be more precisely followed when they are deployed in actual software to be used in real-world scenarios.

# Bibliography

[1] N. Adam and J. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Computing Surveys*, 21(4), 1989.

[2] C. Aggarwal. On k-anonymity and the curse of dimensionality. In *Proc. 31st International Conference on Very Large Databases (VLDB)*, pages 901–909, 2005.

[3] G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the k-th ranked element. In *Proc. Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 40–55. Springer, 2004.

[4] D. Agrawal and C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *Proc. 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 247–255, 2001.

[5] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *Proc. 23rd ACM International Conference on Management of Data (SIGMOD)*, pages 86–97, 2003.

[6] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *Proc. 20th ACM International Conference on Management of Data (SIGMOD)*, pages 439–450, 2000.

[7] A. Asuncion and D.J. Newman. UCI machine learning repository. `http://archive.ics.uci.edu/ml/`, 2007.

[8] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Proc. 4th Theory of Cryptography Conference (TCC)*, volume 4392 of *LNCS*, pages 137–156. Springer, 2007.

[9] M. Bawa, R. Bayardo, and R. Agrawal. Privacy-preserving indexing of documents on the network. In *Proc. 29th International Conference on Very Large Databases (VLDB)*, pages 922–933, 2003.

[10] R. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *Proc. 21st IEEE International Conference on Data Engineering (ICDE)*, pages 217–228, 2005.

[11] D. Beaver. Foundations of secure interactive computing. In *Proc. Advances in Cryptology – CRYPTO 1991*, volume 576 of *LNCS*, pages 377–391, 1992.

[12] M. Bellare and P. Rogaway. Introduction to modern cryptography, 2005. URL: `http://www-cse.ucsd.edu/users/mihir/cse207/classnotes.html`.

[13] I. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Proc. Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 515–529, 2004.

[14] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: the SuLQ framework. In *Proc. 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 128–138, 2005.

[15] X. Boyen, Q. Mei, and B. Waters. Direct chosen ciphertext security from identity-based techniques. In *Proc. 12th ACM Conference on Computer and Communications Security (CCS)*, pages 320–329, 2005.

[16] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[17] J. Brickell, D. E. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *Proc. 14th ACM Conference on Computer and Communications Security (CCS)*, pages 498–507, 2007.

[18] J. Brickell and V. Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *Proc. Advances in Cryptology – ASIACRYPT 2005*, volume 3778 of *LNCS*, pages 236–252, 2005.

[19] J. Brickell and V. Shmatikov. Efficient anonymity-preserving data collection. In *Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 76–85, 2006.

[20] J. Brickell and V. Shmatikov. The cost of privacy: Destruction of data-mining utility in anonymized data publishing. In *Proc. 14th ACM*

*SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 70–78, 2008.

[21] J. Brickell and V. Shmatikov. Privacy-preserving classifier learning. In *Proc. 13th International Conference on Financial Cryptography and Data Security*, 2009.

[22] J-W. Byun, Y. Sohn, E. Bertino, and N. Li. Secure anonymization for incremental datasets. In *Proc. 3rd VLDB Workshop on Secure Data Management*, 2006.

[23] R. Canetti. Security and composition of multiparty cryptograpic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[24] R. Canetti, Y. Ishai, R. Kumar, M. Reiter, R. Rubinfeld, and R. Wright. Selective private function evaluation with applications to private statistics. In *Proc. 20th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 293–304, 2001.

[25] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45:965–981, 1998.

[26] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, and P. Samarati. k-anonymity. In *Secure Data Management in Decentralized Systems*. Springer, 2007.

[27] C. Clifton, M. Kantarcioglou, J. Vaidya, X. Lin, and M. Zhu. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations*, 4(2):28–34, 2002.

[28] Federal Trade Commission. Choicepoint settles data security breach charges. `http://www.ftc.gov/opa/2006/01/choicepoint.shtm`, 2005.

[29] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[30] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proc. Advances in Cryptology – CRYPTO 1998*, volume 1462 of *LNCS*, pages 13–25. Springer, 1998.

[31] T. Dalenius. Finding a needle in a haystack—or identifying anonymous census record. *Journal of Official Statistics*, 2(3):329–336, 1986.

[32] J. Davis, J. Ha, C. Rossbach, H. Ramadan, and E. Witchel. Cost-sensitive decision tree learning for forensic classification. In *Proc. 17th European Conference on Machine Learning (ECML)*, volume 4212 of *LNCS*, pages 622–629. Springer, 2006.

[33] D. E. Denning. Secure statistical databases with random sample queries. *ACM Transactions on Database Systems*, 5(3):291–315, 1980.

[34] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proc. 13th USENIX Security Symposium*, pages 303–320, 2004.

[35] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proc. 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 202–210, 2003.

[36] D. Dobkin, A. K. Jones, and R. J. Lipton. Secure databases: protection against user influence. *ACM Transactions on Database Systems*, 4(1):97–106, 1979.

[37] W. Du and Z. Zhan. Using randomized response techniques for privacy-preserving data mining. In *Proc. 9th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 505–516, 2003.

[38] C. Dwork. Differential privacy. In *Proc. 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4052 of *LNCS*, pages 1–12. Springer, 2006.

[39] C. Dwork. An ad omnia approach to defining and achieving private data analysis. In *Proc. 1st ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD (PinKDD)*, volume 4890 of *LNCS*, pages 1–13. Springer, 2007.

[40] C. Dwork. Differential privacy: A survey of results. In *Proc. 5th International Conference on Theory and Applications of Models of Computation (TAMC)*, volume 4978 of *LNCS*, pages 1–19. Springer, 2008.

[41] C. Dwork, F. Mcsherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *LNCS*, pages 265–284. Springer, 2006.

[42] C. Dwork, M. Naor, O. Reingold, G. Rothblum, and S. Vadhan. On the complexity of differentially private data release. In *Proc. 41st ACM Symposium on Theory of Computing (STOC)*, 2009.

[43] C. Dwork and K. Nissim. Privacy-preserving data mining on vertically partitioned databases. In *Proc. Advances in Cryptology – CRYPTO 2004*, volume 3152 of *LNCS*, pages 528–544, 2004.

[44] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proc. 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 211–222, 2003.

[45] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. *Information Systems*, 29(4):343–364, 2004.

[46] J. Feigenbaum, Y. Ishai, T. Malkin, K. Nissim, M. Strauss, and R. Wright. Secure multiparty computation of approximations. In *Proc. 28th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2076 of *LNCS*, pages 927–938, 2001.

[47] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean. Secure computation of surveys. In *Proc. EU Workshop on Secure Multiparty Protocols*, 2004.

[48] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Proc. Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19, 2004.

[49] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. *Journal of Cryptology*, 17(2):81–104, 2004.

[50] B. Fung, K. Wang, and P. Yu. Top-down specialization for information and privacy preservation. In *Proc. 21st IEEE International Conference on Data Engineering (ICDE)*, pages 205–216, 2005.

[51] Gateway. System management services agreement. `http://www.gateway.com/about/legal/warranties/20461r10.pdf`, 1999.

[52] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Sciences*, 60(3):592–629, 2000.

[53] E-J. Goh, L. Kruger, D. Boneh, and S. Jha. Secure function evaluation with ordered binary decision diagrams. In *Proc. 13th ACM Conference on Computer and Communications Security (CCS)*, pages 410–420, 2006.

[54] O. Goldreich. *Foundations of Cryptography: Volume I (Basic Tools)*. Cambridge University Press, 2001.

[55] O. Goldreich. *Foundations of Cryptography: Volume II (Basic Applications)*. Cambridge University Press, 2004.

[56] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. 19th ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[57] P. Golle, S. Zhong, D. Boneh, M. Jakobsson, and A. Juels. Optimistic mixing for exit-polls. In *Proc. Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 451–465. Springer, 2002.

[58] V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *Proc. Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 289–306. Springer, 2008.

[59] J. Groth. A verifiable secret shuffle of homomorphic encryptions. In *Proc. 6th International Workshop on Theory and Practice in Public Key Cryptography (PKC)*, volume 2567 of *LNCS*, pages 145–160. Springer, 2003.

[60] J. Ha, C. Rossbach, J. Davis, I. Roy, H. Ramadan, D. Porter, D. Chen, and E. Witchel. Improved error reporting for software that uses black box components. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 101–111, 2007.

[61] K. Hafner. And if you liked the movie, a Netflix contest may reward you handsomely. New York Times, Oct 2 2006.

[62] S. Hansell. AOL removes search data on vast group of web users. New York Times, Aug 8 2006.

[63] V. Iyengar. Transforming data to satisfy privacy constraints. In *Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 279–288, 2002.

[64] G. Jagannathan and R. Wright. Privacy-preserving data imputation. In *Proc. 6th IEEE International Conference on Data Mining – Workshops (ICDMW)*, pages 535–540, 2006.

[65] G. Jagannathan and R. Wright. Private inference control for aggregate database queries. In *Proc. 7th IEEE International Conference on Data Mining – Workshops (ICDMW)*, pages 711–716, 2007.

[66] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Proc. Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 97–114, 2007.

[67] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, July 2002.

[68] H. Kargupta, S. Datta, Q. Wang, and K. Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *Proc. 3rd IEEE International Conference on Data Mining (ICDM)*, pages 99–106, 2003.

[69] D. Kifer and J. Gehrke. Injecting utility into anonymized datasets. In *Proc. 26th ACM International Conference on Management of Data (SIGMOD)*, pages 217–228, 2006.

[70] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th ACM Symposium on Theory of Computing (STOC)*, pages 20–31. ACM, 1988.

[71] L. Kissner and D. Song. Privacy-preserving set operations. In *Proc. Advances in Cryptology – CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, 2005.

[72] L. Kruger. Sfe-tools. `http://pages.cs.wisc.edu/~lpkruger/sfe/`, 2008.

[73] D. Lambert. Measures of disclosure risk and harm. *Journal of Official Statistics*, 9:313–331, 1993.

[74] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proc. 25th ACM International Conference on Management of Data (SIGMOD)*, pages 49–60, 2005.

[75] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proc. 22nd IEEE International Conference on Data Engineering (ICDE)*, page 25, 2006.

[76] K. LeFevre, D. DeWitt, and R. Ramakrishnan. Workload-aware anonymization. In *Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 277–286, 2006.

[77] N. Li, T. Li, and S. Venkatasubramanian. $t$-closeness: Privacy beyond $k$-anonymity and $\ell$-diversity. In *Proc. 22nd IEEE International Conference on Data Engineering (ICDE)*, pages 106–115, 2007.

[78] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[79] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Proc. Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.

[80] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[81] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. $\ell$-diversity: Privacy beyond $k$-anonymity. In *Proc. 22nd IEEE International Conference on Data Engineering (ICDE)*, pages 24–35, 2006.

[82] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. In *Proc. 13th USENIX Security Symposium*, pages 287–302, 2004.

[83] D. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J. Halpern. Worst-case background knowledge for privacy-preserving data publishing. In *Proc. 23rd IEEE International Conference on Data Engineering (ICDE)*, pages 126–135, 2007.

[84] G. McGraw and J. Viega. Making your software behave: Security by obscurity. `http://www.ibm.com/developerworks/java/library/s-obs.html`, 2000.

[85] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *Proc. 48th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 94–103, 2007.

[86] Microsoft. Privacy statement for the Microsoft error reporting service. `http://oca.microsoft.com/en/dcp20.asp`, 2006.

[87] Microsoft. Reporting and solving computer problems. `http://windowshelp.microsoft.com/Windows/en-US/Help/d97ba15e-9806-4ff3-8ead-71b8d62123fe1033.mspx`, 2006.

[88] Microsoft. How to: Configure microsoft error reporting. `http://msdn2.microsoft.com/en-us/library/bb219076.aspx`, 2007.

[89] M. Naor and K. Nissim. Communication preserving protocols for secure function evaluation. In *Proc. 33rd ACM Symposium on Theory of Computing (STOC)*, pages 590–599, 2001.

[90] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proc. 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 448–457, 2001.

[91] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proc. 1st ACM Conference on Electronic Commerce*, pages 129–139, 1999.

[92] R. Naraine. Dr. Watson's Longhorn makeover raises eyebrows. `http://www.eweek.com/article2/0,1759,1822142,00.asp`, 2005.

[93] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Proc. 29th IEEE Symposium on Security and Privacy (SP)*, pages 111–125, 2008.

[94] A. Neff. Verifiable mixing (shuffling) of ElGamal pairs. `http://www.votehere.net/vhti/documentation/egshuf.pdf`.

[95] M. Nergiz, M. Atzori, and C. Clifton. Hiding the presence of individuals from shared databases. In *Proc. 27th ACM International Conference on Management of Data (SIGMOD)*, pages 665–676, 2007.

[96] M. Nergiz and C. Clifton. Thoughts on k-anonymization. *Data and Knowledge Engineering*, 63(3):622–645, 2007.

[97] M. Nergiz, C. Clifton, and A. Nergiz. Multirelational k-anonymity. In *Proc. 23rd IEEE International Conference on Data Engineering (ICDE)*, pages 1417–1421, 2007.

[98] Netflix. Netflix Prize. `http://www.netflixprize.com/`, 2006.

[99] A. Øhrn and L. Ohno-Machado. Using boolean reasoning to anonymize databases. *Artificial Intelligence in Medicine*, 15(3):235–254, 1999.

[100] Oracle. Oracle sues SAP. `http://www.oracle.com/sapsuit/index.html`, 2007.

[101] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. Advances in Cryptology – EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 223–238, 1999.

[102] H. Park and K. Shim. Approximate algorithms for k-anonymity. In *Proc. 27th ACM International Conference on Management of Data (SIGMOD)*, pages 67–78, 2007.

[103] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[104] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[105] Vibhor Rastogi, Dan Suciu, and Sungho Hong. The boundary between privacy and utility in data publishing. In *Proc. 33rd International Conference on Very Large Databases (VLDB)*, pages 531–542, 2007.

[106] The Register. Acxiom database hacker jailed for 8 years. `http://www.theregister.co.uk/2006/02/23/acxiom_spam_hack_sentencing/`, 2006.

[107] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[108] T. Sander, A. Young, and M. Yung. Non-interactive CryptoComputing for NC1. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 554–566, 1999.

[109] B. Stone. A lively market, legal and not, for software bugs. New York Times, Jan 30 2007.

[110] L. Sweeney. Weaving technology and policy together to maintain confidentiality. *Journal of Law, Medicine and Ethics*, 25(2–3):98–110, 1997.

[111] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):571–588, 2002.

[112] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[113] P. Syverson, D. Goldschlag, and M. Reed. Anonymous connections and onion routing. In *Proc. 18th IEEE Symposium on Security and Privacy (SP)*, pages 44–54, 1997.

[114] T. Truta and B. Vinay. Privacy protection: p-sensitive k-anonymity property. In *Proc. 22nd IEEE International Conference on Data Engineering – Workshops (ICDEW)*, page 94, 2006.

[115] J. Vaidya and C. Clifton. Privacy-preserving association rule mining in vertically partitioned data. In *Proc. 8th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 639–644, 2002.

[116] J. Vaidya, C. Clifton, M. Kantarcioglu, and A.S. Patterson. Privacy-preserving decision trees over vertically partitioned data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(3):1–27, 2008.

[117] J. Vaidya, M. Kantarcioglu, and C. Clifton. Privacy-preserving Naive Bayes classification. *The VLDB Journal*, 17(4), 2008.

[118] V. Verykios, E. Bertino, I. Fovino, L. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1):50–57, 2004.

[119] K. Wang and B. Fung. Anonymizing sequential releases. In *Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 414–423, 2006.

[120] K. Wang, B. Fung, and P. Yu. Template-based privacy preservation in classification problems. In *Proc. 5th IEEE International Conference on Data Mining (ICDM)*, pages 466–473, 2005.

[121] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *The American Statistical Association*, 60(309):63–69, 1965.

[122] J. Weideman. Automated problem reports. `https://wiki.ubuntu.com/AutomatedProblemReports`, 2006.

[123] D. Wikström. Four practical attacks for "optimistic mixing for exit-polls". Technical Report T2003-04, Swedish Institute of Computer Sciences (SICS), 2003.

[124] D. Woodruff and J. Staddon. Private inference control. In *Proc. 11th ACM Conference on Computer and Communications Security (CCS)*, pages 188–197, 2004.

[125] D. P. Woodruff. Revisiting the efficiency of malicious two-party computation. In *Proc. Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 79–96. Springer, 2007.

[126] R. Wright and Z. Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *Proc. 10th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 713–718, 2004.

[127] X. Xiao and T. Tao. M-invariance: towards privacy preserving republication of dynamic datasets. In *Proc. 27th ACM International Conference on Management of Data (SIGMOD)*, pages 689–700, 2007.

[128] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *Proc. 32nd International Conference on Very Large Databases (VLDB)*, pages 139–150, 2006.

[129] Z. Yang, S. Zhong, and R. Wright. Anonymity-preserving data collection. In *Proc. 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD)*, pages 334–343, 2005.

[130] Z. Yang, S. Zhong, and R. Wright. Privacy-preserving classification of customer data without loss of accuracy. In *Proc. 5th SIAM International Conference on Data Mining (SDM)*, pages 21–23, 2005.

[131] Z. Yang, S. Zhong, and R. Wright. Towards privacy preserving model selection. In *Proc. 1st ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD (PinKDD)*, volume 4890 of *LNCS*, pages 138–152. Springer, 2007.

[132] A. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.

[133] L. Zhang, S. Jajodia, and A. Brodsky. Information disclosure under realistic assumptions: privacy versus optimality. In *Proc. 14th ACM Conference on Computer and Communications Security (CCS)*, pages 573–583, 2007.

# Vita

Justin Brickell was born in Columbus, Ohio in 1980. He graduated from Albuquerque Academy in Albuquerque, New Mexico, in 1999, and received the degrees of Bachelor of Arts (in Mathematics) and Bachelor of Science (in Computer Science) from Rice University in Houston, Texas, in 2003. He enrolled in the Ph.D. program at the University of Texas at Austin in 2003, and received the degree of Master of Science in Computer Sciences in 2006.

Permanent address: 1370 University Ave Unit 403
Berkeley, CA 94702

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.