Abstract

Implementation and Evaluation of Privacy-Preserving Protocols

Felipe Saint-Jean Antonijevic

2010

Computers and networks have become essential tools for many daily activities, including commerce, education, social interaction, and political engagement. Massive amounts of sensitive data are transmitted over networks and stored in web-accessible databases, and thus privacy of these data has become a top priority. Decades of research in cryptography and security have resulted in an elegant theory that shows, in principle, how to perform almost any computational task in a privacy-preserving manner, but there is a wide gap between theory and practice: Currently, most people and organizations have no straightforward way to safeguard the privacy of their sensitive data in realistic networked environments.

In the interest of bridging this gap, we design and implement protocols to enhance privacy in four scenarios: web search, security-alert sharing, database querying, and survey computation. We evaluate our protocols and implementations with respect to various natural criteria, including effectiveness, efficiency, and usability. Our principle contributions include:

◦ We introduce PWS, a Firefox plug-in that enhances privacy in web search by minimizing the information that is sent from the user's machine to the search engine. PWS protects users against attacks that involve active components and timing information, to which more general web-browsing privacy tools are vulnerable. In a study of ease of installation and use, PWS compares favorably with the widely used TPTV bundle.

◦ We design, implement, and analyze a threshold-union protocol that allows networks to share security-alert data in a consensus-private manner, *i.e.*, one in which an anomaly discovered at one network is only revealed if it is discovered at $t-1$ other networks, where t is the threshold. Our protocol achieves entropic security, accommodates transient contributors, and is significantly more scalable than the earlier protocol of Kissner and Song.

◦ We implement a computationally secure protocol for symmetric, private information retrieval

that uses oblivious transfer in an essential way. Our implementation is fast enough for medium-sized databases but not for large databases.

◦ Motivated by the CRA Taulbee survey of faculty salaries, we use the FairPlay platform of Malkhi *et al.* to build a highly usable system for privacy-preserving survey computation.

Implementation and Evaluation of Privacy-Preserving Protocols

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Felipe Saint-Jean Antonijevic

Dissertation Director: Joan Feigenbaum

December 2010

Contents

List of Figures

List of Tables

Dedication and Acknowledgments

This thesis is dedicated to my wife Carolina, who has always been there for me. I will always be there for you.

I am indebted to my advisor Joan Feigenbaum for her invaluable advice, support, and encouragement.

The research presented in this thesis was supported by NSF grants 0207399, 0331548, and 0534052, ONR grant N00014-04-1-0725, AFOSR grant FA8750-07-2-0031, and a Kern Fellowship.

Chapter 1

Introduction

Computers and networks have become essential tools for many daily activities, including commerce, education, social interaction, and political engagement. Massive amounts of sensitive data about people and organizations are transmitted over networks and stored in web-accessible databases, and thus maintaining the privacy of these data has become a top priority. Decades of research in cryptography and security have resulted in an elegant theory that shows, in principle, how to perform almost any computational task in a privacy-preserving manner, but there is a wide gap between theory and practice: Currently, most people and organizations have no straightforward way to safeguard the privacy of their sensitive data in realistic networked environments.

In the interest of bridging this gap, we design and implement protocols to enhance privacy in four important tasks: web search, security-alert sharing, database querying, and survey computation. We also evaluate our protocols and implementations with respect to various natural criteria, including effectiveness, efficiency, and usability. We briefly summarize our principle contributions here and elaborate on them in Chapters 2 through 6.

PWS: A Firefox plug-in for private web search

Web search is currently a source of growing concern about personal privacy. It is an essential and central part of most users' activity online and therefore one through which a significant amount of personal information may be revealed. To help search-engine users protect their privacy, we have designed and implemented Private Web Search (PWS), a usable client-side tool that minimizes the information that users reveal to a search engine. Our tool is aimed specifically at *search privacy* and thus is able to protect users against attacks that involve active components and timing information, to which more general Web-browsing privacy tools (including the widely used TPTV bundle – Tor [61], Privoxy [48], Torbutton [62], and Vidalia [69]) are vulnerable. PWS is a Firefox plugin that functions as an HTTP proxy and as a client for the Tor anonymity network [61]. It configures Firefox so that search queries executed from the PWS search box are routed through the HTTP proxy and Tor client, filtering potentially sensitive or identifying components of the request and response.

We present the design and implementation of PWS in Chapter 2.

A user study of PWS and the TPTV bundle

We turn next to a user study that compares PWS, TPTV, and Google with no privacy ehancements. Study subjects had significantly more difficulty using the TPTV bundle than they did simply using "plain" Google. Users of PWS did much better. In an attempt to understand the reasons for adoption of web-privacy technology (or the lack thereof), we also surveyed the study participants about their level of concern about web privacy and their reasons for using or not using brower-based privacy tools. Most users expressed concern about privacy and willingness to take action to address it, but they also said that they would do not use Firefox extensions

such as TPTV or PWS because of the latency that these extensions introduce to the search process.

We present this user study in Chapter 3.

Consensus-private sharing of network-security alerts

Detection of viruses, denial-of-service attacks, and other unwanted traffic is an important security challenge. Many sites use anomaly-detection systems that generate alerts when "suspicious-looking" traffic arrives. It would be useful if sites pooled their alert data so that network-wide threats could be distinguished from traffic that merely looks suspicious locally. Because locally generated alert data are sensitive, sites will not share them unless privacy is addressed.

We propose to address it by using the threshold-union function of Kissner and Song [33]. The *t-threshold union* A of alert sets $A_1, \ldots, A_n$ contains all a that occur in at least t distinct sets $A_{j_1}, \ldots, A_{j_t}$. Our premise is that, for sufficiently large t, the t-threshold union would not be considered sensitive, because, by definition, it cannot depend intimately on the proprietary state of any one site. We provide a protocol for t-threshold union that is consensus-private: It reveals all of A but hides $(A_1 \cup \cdots \cup A_n) \setminus A$.

Our protocol is more practical and scalable than previously known consensus-private protocols for threshold union. The cost of increased efficiency is the need for a stronger assumption about the probability distributions on the sites' private alerts.

We present our threshold-union protocol in Chapter 4.

Implementation of a PIR (private information retrieval) protocol

In the *Private Information Retrieval* problem (PIR), one party owns a database, and the other wants to query it without revealing the query specifications, *e.g.*, to retrieve from an n-record database $\{r_1, \ldots, r_n\}$ the i^{th} record r_i without revealing

the index i to the database owner. Since the PIR problem was first put forth in this form [7], many variations and technical approaches have been proposed [25]; in particular, a great deal of attention has been focused on *symmetric* private information retrieval (SPIR) protocols, in which the goal is not only to hide the query from the database owner but also to hide from the querier all information about records in the database that do not match his query. The question of whether PIR techniques could be useful in practice is as yet unanswered (and, in fact, largely unaddressed). We have implemented a computationally secure SPIR protocol; our experience with this implementation indicates that SPIR could be useful for medium-sized databases but not for large ones.

We present our PIR implementation in Chapter 5.

Privacy-preserving survey computation

Motivated by the privacy concerns inherent in the CRA Taulbee survey of faculty salaries in North American Computer Science departments [9], we design, implement, and evaluate a system for privacy-preserving survey computation. Our point of departure is the *FairPlay* package [39] for secure, two-party function evaluation. To achieve the desired efficiency, we used FairPlay's runtime system but not its compiler; instead, we built a customized garbled-circuit generator for sorting networks. Our system is highly usable, requiring survey participants simply to fill in web forms exactly as they would in a survey application that did not protect data privacy.

We present our work on privacy-preserving survey computation in Chapter 6.

Note that our goals in this work do *not* include mathematical formulation of new privacy definitions or new adversary models. Rather, we attempt to shed light on the steps that would need to be taken to apply existing "provably secure" protocols to real-world problems.

Chapter 2

Private Web Search[1]

2.1 Introduction

The August 2006 release by AOL of the search queries of approximately 650,000 people [17] served as an alert to the privacy threat of online searching. Although the users' IP addresses were replaced with meaningless numbers, it was easy in many cases for a member of the general public to identify a user from his queries. The search-engine company itself has even greater power to identify users. This is worrisome, because queries can be very revealing, and the number of queries that search engines receive is growing as they improve and expand their web databases. Search-engine companies are strongly motivated to collect and analyze these data, because their business model is based on extracting user information in order to better target online advertising.

The Web-search scenario is also a good one in which to have a focused discussion about privacy. It has a few properties that are extremely important from the user's point of view, *i.e.*, with respect to actions the user can take to control what he

[1]This chapter is based on [54].

reveals about himself. First, search services are widely used, and thus there is hope of hiding in the crowd. Second, because of the large number of users, a concrete, widely available tool for enhancing privacy might produce useful feedback. Third, it is a point of connection among most web activities; so the privacy concerns are larger than in more specific web services.

Private Web Search (PWS) is a Firefox plugin that protects the user's privacy by minimizing the information sent to the search engine, while still getting the query result. This is done by filtering the request and response and by routing them through an anonymity network. The user is thus protected from information leaks that might lead to his identification.

2.2 Problem statement

Search-engine queries can reveal a great deal about a user. The query terms themselves can include clearly identifying information such as a user's name, Social Security Number, and location. They may also indicate things about a user's work, family, interests, and future plans. Other aspects of the search request, such as the IP address, the HTTP headers, and the time of day, also let search engines learn things about the user, such as his ISP, browser, and activity at a specific time. Clearly, many users would like to keep this kind of thing private. Privacy must, however, be achieved while still providing users with the search functionality. A trivial way to protect privacy would be to send no queries at all, but this is unacceptable.

Before we make our notion of privacy more precise, consider several scenarios in which a user's search queries are used to try to learn things about him:

1. The search-engine company runs a large-scale profiling operation in which it tries to learn as much as possible about its users. The engine could link queries

and build user profiles under the assumption that queries done on the same day from the same IP address come from the same user. These profiles could be combined with information found online, such as personal web pages and government records, to learn things such as the users' names and addresses.

2. The search-engine company is more focused and monitors queries for terms of interest. These could include things like subjects of national-security interest to the government or products of partner companies. Once a term of interest is encountered, it could, as before, be linked to other queries issued around the same time and from the same IP address, as well as with online sources of information. If the terms of interest were selective enough and the interested party motivated enough, the profiles could also be compared to and combined with less available sources of information such as logs from the user's ISP or public records that must be retrieved physically.

3. The adversary wants to learn the queries of a specific user. Perhaps an employee with access to the data is curious about a celebrity, or law enforcement is gathering data in a criminal investigation. In this case, the adversary has significant background knowledge about the user to start with. The adversary might, for example, know where the user was at a certain time or what his ISP is; perhaps the adversary can guess the query terms that the user is likely to use, such as his name or something related to his work. It is easy to see how this background information can help the adversary determine the queries that were issued by the user.

In all of these situations, the privacy concern arises as the search engine becomes able to make good guesses about the source of some search queries. The engine is aided in this task by knowledge of user behavior - some that it starts with and some

that it develops as it examines the queries. We don't have much hope of preventing an adversary from guessing the source of queries that are likely to come from only a few users: full names, addresses, credit-card numbers, etc. Therefore, our privacy goal will be to prevent an adversary from improving its guess too much after observing incoming queries, while still providing users with the search results that they want.

To state the privacy issue more concretely, assume that there is some set of users U and that the adversary knows the size of U. We can model web search as a probabilistic process. Let there be some probability distribution on the search queries that the users will make in a given time period. Our adversary has prior knowledge about search queries made in some period of time in the form of an estimate of the probability distribution over sets of queries. He gets some information about the queries that were performed in the form of the value of a random variable that represents his observations. From this he can infer a conditional distribution on which queries were performed and by whom. We want to minimize the difference between the prior distribution and this posterior distribution. In particular, we don't want to increase by too much the probability that a particular user issued a particular query.

We won't develop this model of the problem any further in this chapter; nor will we attempt to precisely express and analyze PWS or other solutions in it. However, we will use it to understand how different approaches protect privacy. Moreover, this view of privacy illustrates how the problem of private web search relates to other privacy problems that have been studied.

There are several practical tools [23, 48, 65] that offer ways to hide major clues (*e.g.* IP address) to the user's identity. However, for the most part they do not address more subtle attacks such as Flash cookies and cache timing [21, 31]. Also, none of these tools is convenient and comprehensive. We want to provide a tool that is easy to use and is effective at protecting users' privacy during web search.

2.3 Related Work

2.3.1 Current approaches

One straightforward way to protect privacy in web searching is to use an anonymizing proxy. Lists of freely accessible proxies are available online. Using these hides the true source IP address. However, because all queries are sent through the same proxy they can easily be linked together. Also, the adversary need only obtain logs from the proxy to determine the true source.

These concerns can be addressed by using the anonymity network Tor [61], which is essentially a network of anonymizing proxies. The source of the connection to the search engine is rotated periodically among the routers in Tor. Also, connections are routed through several Tor routers and encrypted in such a way that logs from all are necessary to determine the true source.

Still, the HTTP request itself might release information about the source, *e.g.*, through cookies or the User-Agent header. Also, the HTTP response might include ways to get the client to reveal itself, such as JavaScript that communicates with the engine. A filtering HTTP proxy, such as Privoxy [48], can eliminate some of these possibilities, but it is a general tool for all web browsing that does not include sufficient filtering for web-search results. In particular, the search engine can employ techniques such as redirects in the search results and cache-timing attacks [21, 31].

This solution may also be somewhat difficult for users to install and configure. Browser plugins, such as the FireFox plugins FoxTor [23] and TorButton [62], can make this easier. Even with these tools, if the user doesn't want to run all HTTP requests over the slower Tor network, he must go through the effort of manually enabling and disabling the use of Tor.

The TrackMeNot[65] tool uses a different approach. It attempts to protect the

user's privacy by issuing computer-generated random queries. The objective is to make it hard for the search engine to distinguish real user queries from the computer-generated "cover traffic." Users can be identified by IP address, but what they search for is obscured to some extent by noise. TrackMeNot has not been formally analyzed, however, and it is not clear how indistinguishable one can actually make the false queries from real ones. Real queries are often semantically related in subtle ways and may include very specific and identifying terms (*e.g.*, addresses and names). This scheme also adds undesirable extra load on the search engine.

2.3.2 Related privacy research

Two well known problems in the privacy literature have significant similarities to private web search: privacy-preserving data publishing [68] and private information retrieval [25].

Privacy-preserving data publishing is the problem of making a database of personal information available to the public for data mining without revealing private information about individuals. Census officials, for example, may want to provide census data so that researchers can learn general things about the population. However, they don't want to expose any individual's private information, such as his or her income. Some approaches to this problem include generalizing identifying fields [56, 60, 38], adding random noise to the entries in the database [5, 18], and randomly adding and deleting entries [49].

Private web searching can be viewed as an instance of this problem by taking the database to be the set of search queries, the publisher to be the users, and the public to be the search engine. Solutions to privacy-preserving data publishing therefore suggest solutions to private web search. Hiding the source IP address and normalizing the HTTP headers of a web request, for example, can be viewed as an

application of the generalization technique. PWS adds random noise to the response time of a query by sending it over the Tor network. Using Tor perturbs the network latency, making it harder to identify users based on their network round-trip time. This is because the network latency will depend on the randomly chosen Tor path. We are prevented by our functionality requirement from deleting queries, but adding random queries is exactly the approach taken by the TrackMeNot utility [65].

Web search differs from data publishing in several ways that affect the ability to transfer solutions between the two. First, the data in web search are being "published" by many users who are unknown to one another. We want to avoid any solution that requires coordination among the users, such as k-anonymous generalization [56, 60]. Also, web search has a limited functionality requirement - we must obtain search results. Therefore we can freely modify any part of the request other than the search terms without being concerned that it might affect the utility of the data for data mining. Finally, because we must obtain accurate search results, we cannot in general add noise to the query terms.

In the Private Information Retrieval problem (PIR) [25], which we consider in Chapter 5, a user wants to query a database without revealing the query. It isn't hard to see that solving this problem would solve the web search problem. One simple PIR solution is for the database owner to send a copy of the database to the user. The size of the database may well be very large, however. Solutions that are information-theoretically secure and have lower communication requirements [7] involve querying copies of the database. Single-copy solutions with asymptotically low communication based on computational-hardness assumptions also exist [34].

The problem with applying PIR schemes to private web search is that search databases are huge. Replication for privacy purposes would be very costly. The single-database PIR schemes just aren't fast enough. Their response algorithms

must touch every piece of the data when computing a response, or the adversary can determine that some entries were not queried.

2.4 On anonymity networks

When a user establishes a connection with a server and is concerned about the misuse of the personal information that the server will gather during the connection, there are a couple of approaches he can take. He can understand the server's privacy policy and trust the server to enforce it, or he can remain anonymous. Of course not all services can be accessed anonymously, but this approach should work for Web searching.

We built a client-side tool because we do not trust servers to enforce reasonable privacy policies. Indeed, a complaint in the AOL case has been lodged with the FTC arguing that AOL violated its own privacy policy [17]. That being the case, we want users to remain anonymous. A key step in realizing this is the use of anonymity networks [61, 32], and in particular the Tor onion-routing network, to obscure the source of the connection. Onion routing [50] uses a network of routers to forward messages in a way that breaks the link between incoming and outgoing traffic. This is done by layering encryption so that each router knows the previous and the next hops but nothing more. This kind of network provides practical and robust anonymity for Internet traffic and thus is very useful for our project. Tor [14] is a widely used implementation of onion routing. As of January 2007, it consists of over 800 routers and serves an estimated 200,000 users.

We are using Tor for the specific purpose of private web search, and so there may be ways to customize its operation. One that we have implemented is building 2-hop paths instead of 3-hop paths. The argument for three hops in Tor is that an

adversary that controls a router should not be able to know all of the routers on the circuits it observes. Our adversary is a search-engine company, however, and we assume that it does not try to break the anonymity of the Tor network. Therefore, we can improve speed by removing one hop. We also suggest that Tor-router operators may be more willing to act as exit nodes for the popular search engines. Providing them with exit policies that allow such access could help the performance of our tool.

Tor provides an important part of our solution by hiding the source IP address. However, the search engine can get information about the source of a query in other ways.

2.5 Sensitive information in Web searching and user tracking

There are many sources of potentially identifying information in web search. Table 2.1 show a summery of this information. First, there are IP addresses. The IP address by itself provides a large amount of information about the user. It may provide geographical location, ISP, or institution. Although associating exactly one user with an IP address is not simple, the address certainly narrows the possibilities. Because a user's IP address will, in all likelihood, be the same for a while, it provides strong linkability among queries issued during that time period. There is also timing information at the IP level. The server side of the connection will be able to estimate the round-trip time (RTT) of packets. This will allow the adversary to distinguish between users with RTTs that are far apart.

As seen in [46], at the TCP level, inspection of packets can reveal information about the machines involved in the connection. Uptime (time since system boot), operating system, and other properties of the connection can be estimated with high

	Level	Identifying information	Solution
1	TCP/IP		Tor
		IP address	
		Institution or ISP	
		Operating system	
		Uptime	
		Timing (RTT)	
2	HTTP Headers		HTTP filter
		Cookies	
		Operating system make and version	
		Browser make and version	
		Encoding and language	
3	HTML		HTML filter
		JavaScript	
		Timing (web timing attacks)	
4	Application		Open problem
		Query terms	
		Time of day of the query	
5	Active components		HTML filter
		...	
		...	
		...	

Table 2.1: Gray items (3 and 5) are new to PWS and provide protection not provided by Tor+Privoxy. The ...s in item 5 are meant to indicate that the type of information captured by active components is application specific and thus has no succinct description.

accuracy by passive inspection of the network traffic.

Then, there is the HTTP header, in which there is a lot of information that allows tracking. Cookies can usually uniquely identify the user, and they provide query linkability for even longer periods of time than the IP address. Also, there is a large set of flags and markers that give the search engine specifics about the user's system. Although these are required for the processing of many types of web requests, they are not required for search.

Once the search engine gives a response, the Web browser will interpret and execute all elements in the Web page. Many of these elements are designed to provide the user with a rich experience and will thus initiate further network connections. Each additional connection may implicate privacy. For example, image, frame, and

style tags will generate the download of additional files. Each of these connections must be dealt with in detail.

Beyond HTML tags, there are a variety of active components that can be embedded in the web page. These active components are used in general to improve the user's experience and enhance user interfaces, but they can also reveal private information about the user. For example, it is common practice for search engines to use JavaScript in order to get feedback about the URL selected by the user. Most of the active components execute within a constrained environment, but they are generally able to transmit to the server information specific to the user. In addition to JavaScript, we must deal with Flash, ActiveX, Java, and a variety of plugins. These active components could, in principle, be used to fingerprint the user's machine, potentially identifying the user.

The search engine can also use several web-timing attacks [21, 31] to distinguish among users. The contents of a user's web cache can be queried by sandwiching a request for some page between queries to the search engine's own content in the HTML response. By measuring the time between the requests for its own content, it can determine whether the page is in the user's cache. The contents of the user's DNS cache can be similarly queried. The search engine can potentially use this technique to read and write unique signatures in user caches.

The last piece of information transmitted when searching the Web is the search query itself. Each query gives some information about the user. In other words, a given query could be issued by a subset of the user base but probably not by all users. The problem grows as queries are linkable. Each query reduces the set of possible users that might have issued the queries, and, once the set of queries gets large enough, they can be related to a user or a small set of users. This means that, in order to achieve our main objective, we need to work towards reducing the

linkability of queries to each other as well as the linkability of queries to users. It should be hard for the search engine to determine that a large set of queries was issued by the same user.

Table 2.1 summarizes the information that is revealed by a general Web transaction. 1 and 2 are very standard and mostly independent of the application or website. That is why 1 and 2 can be addressed by general Web-privacy tools like Tor+Privoxy. We can't expect to hide 4 without the search engine's cooperation and maintain functionality and usability. But what about 3 and 5? The information that is revealed at those levels is application-specific; thus, it is not possible to delete this information without considering the application semantics and expect to preserve functionality. This issue requires a specific solution for each Web application. PWS is a specific solution for Web search.

2.6 Implementation

2.6.1 General architecture

PWS is a Firefox plugin within which run a Tor client and an HTTP proxy. When the user executes a query, it connects to the HTTP proxy. The proxy filters the HTTP request, then sends it to the search engine over the Tor network. Later the proxy receives the response from Tor, filters the HTML to remove all active components, and gets the answer back to Firefox for display. Table 2.1 shows which PWS modules take care of the various types of information leaks that may occur during search. Right now PWS can only be used with Google [29]; it would be straightforward to extend the plugin to let users select from multiple search engines, and we intend to do so. Also, Google guesses the user's desired language from the IP address of the source, and Tor may send the query from servers around the world. Therefore we currently

send queries to the English language URL (`http://www.google.com/intl/en/`). Allowing users to select a search language slightly reduces anonymity, but it an essential usability feature that would be worth including.

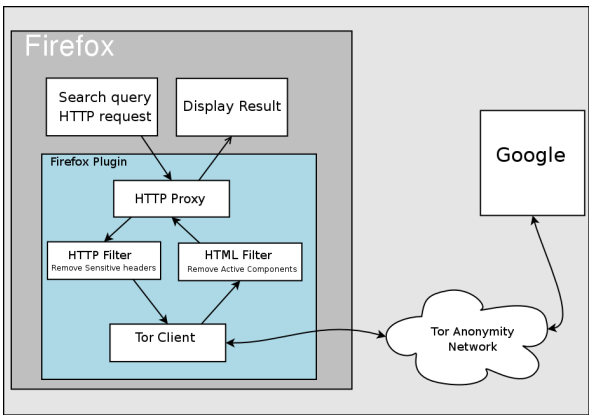


Figure 2.1: PWS general architecture

2.6.2 HTTP filter

The HTTP module's goal is to normalize the HTTP request so that it looks as similar as possible across all PWS users. Query terms will be different, of course, but all protocol specifics of the connection should be removed. A general HTTP header looks like Figure 2.2.

Much of the information in this header is not needed to resolve the query but helps the search engine to identify users. The HTTP filter in PWS makes all headers look something like Figure 2.3.

The only things that change from user to user are the query terms. This is the only module that must be reimplemented to support different search engines.

```
GET /search?hl=en&lr=&q=tennis+tournament&btnG=Search HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.0.7) \\
            Gecko/20060921 Ubuntu/dapper-security Firefox/1.5.0.7
Accept: text/xml,application/xml,application/xhtml+xml,text/html;\\
            q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.google.com/search?hl=en&q=tennis&btnG=Google+Search
Cookie: PREF=ID=71566fe64d9cded5:TM=1165265408:LM=1165265408: \\
         S=b7rWrEz_I8UIXl5U; SID=DQAAAGoAAABj6AMZNxlm1JiSeJcN0jZG [...]
```

Figure 2.2: Sample HTTP-header section

```
GET /search?hl=en&q=tennis+tournament HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0
Accept-Encoding: gzip,deflate
```

Figure 2.3: The result of applying PWS's HTTP filter to the sample in Figure 2.2

2.6.3 Tor client

The Tor client serves two different purposes. First, it makes it hard for the search engine to link a user to a source IP address of a query. Second, it allows us to change that source IP address between queries in order to reduce query linkability. Every query is issued through a different channel and, as long as the query rate is below our channel-rebuild rate, channels are not reused. Therefore the source IP of every query is randomly and indepently selected from the routers in the Tor network (or, more accurately, from the exit nodes).

Our implementation of a Tor client makes heavy use of Jap [32]. The Jap proxy is not solely a Tor client, but it implements one as part of its feature set. The Jap Tor client is written in Java and embedded in the PWS plugin using the Java Firefox Extension code from the SIMILE [57] project. We use paths of length 2, instead of the default 3, in order to improve performance. Still, using PWS is much slower than a direct search, because queries are routed through the Tor network. In our tests, a query was, on average, 20 times slower than a direct connection: 10 seconds vs 0.5 seconds. The variance in response time was much higher, too.

2.6.4 HTML filter

The HTML filter's job is to remove any component that may provide feedback to the search engine. Recall that this is the additional privacy protection that PWS provides over and above the protection provided by Tor+Provoxy. This is done by parsing the response HTML and extracting only the information needed to present an answer to the user. The information extracted is the result description, text abstract, and result URL. These are extracted using regular expressions. Using the extracted text, a new HTML file is built, with all HTML generated by PWS. This has only

the results and no embedded objects. This means that the user only performs one HTTP GET per query, preventing cache timing attacks [21]. All active components such as JavaScript and Flash are removed, so that no extra code is executed.

One alternative approach might be to disable all features in the browser that can initiate additional connections and leak information. We prefer our approach for two reasons. First, it gives better assurance that all active components and undesired labels are removed, even those we did not think about while writing PWS. Second, it gives us control over the functionality. It is possible that disabling some feature will break functionality. By generating new HTML, we can be sure that functionality is preserved.

Because we are protecting only connections to the search engine, we need to be much more careful about additional connections than other tools that filter all connections. Just loading an image is very dangerous if the resulting connection is not filtered. The HTML filter makes sure this does not happens.

Changes in the search engine's HTML code can make the HTML filter fail to produce an output. That means that we need to keep up with the changes. We don't expect the HTML in the response to change too frequently. Also, updating Firefox plugins is a painless process for the user.

2.7 How to use PWS

Usability is an important objective of ours, and so in this section we describe briefly how the plugin works from the user's point of view.

The PWS plugin is distributed as a single .xpi file that, to be installed, needs to be dragged into the plugin installation window. Once it's installed, Firefox will need to be restarted. After that, a new search option will show up among the search

engines in the search box (see Figure 2.4). That search option is labeled “PWS Google search." When the user searches using this option, the query will be routed through the HTTP proxy and the Tor client. The response will take some more time and will look different from the standard Google response, because the HTML was filtered and then displayed. Notice that the user can choose between regular Google queries and PWS just by selecting accordingly from the search box.

Figure 2.4: PWS user interface

2.8 Conclusions

Taken together, search queries can reveal a lot of information about a user. Experience has shown that we cannot rely on the search-engine companies not to release this information or otherwise use it in undesirable ways. Therefore, we suggest that users might want to minimize the information they give to the search engines by preventing them from identifying the source of the queries or even which queries originate from the same user.

Preventing search engines from linking queries requires eliminating all potentially unique information that gets sent to them. We have identified the IP layer, the HTTP request, and the browser’s behavior on the HTML response as likely sources of this information and have designed a tool that prevents all of these sources from leaking. Previous tools deal only with part of the potentially identifying information; so PWS

is a contribution over exiting software.

Our tool, Private Web Search (PWS), is a Firefox plugin that gives users a search box in the browser for private search. Queries are sent over the Tor anonymity network, after the HTTP headers have been normalized, and the results are extracted from the response to prevent active components or timing attacks from identifying the user to the engine. The plugin is easy to install and use (as explained in the next chapter) and works transparently in the background after a request is made, with an increase in the response time as the only major change in the user experience.

There is a very interesting side impact of this project. Search-engine companies, as seen with AOL, want to release data to third parties in order to improve their understanding of web search. The problem is that there is no agreed-upon privacy standard for releasing these data. If we are able to reach agreement that our tool protects individual users' privacy, that by itself will advance the discussion about data-release policies. The search-engine company could simulate the case in which every user uses PWS and thus generate a dataset acceptable for distribution. This may, however, reduce the usefulness of the data for research purposes.

Chapter 3

Private Web Search Usability[2]

3.1 Introduction

PWS and other web-privacy tools that use similar techniques exhibit network effects; that is, the benefit that any single user derives from using them grows with the total number of users. Thus, it is important to ask whether there is widespread interest among users in adopting such tools and, if there is, whether widespread adoption is feasible. For it to be feasible, typical users would have to find the tools easy to use, and they would have to be satisfied with the tools' performance. These are the questions we address in this chapter.

Our main contribution is the presentation and analysis of the results of a study of 41 users at Yale University. The study was designed to discover how well users were able to install PWS, how well they were able to install TPTV (the well known "Tor, Privoxy, Torbutton, Vidalia" bundle for general Web privacy), and how effectively they were able to search using PWS or TPTV (compared to their effectiveness when using Google without any privacy tools). Users were also surveyed about their level

[2]This chapter is based on [53].

of concern about privacy and their reasons for adopting a browser-based web-privacy tool such as PWS or TPTV or for choosing not to adopt such a tool.

Users were divided into three groups. The control group used Google without any additional privacy-enhancing technology; Google was chosen because it is the most widely used search engine and thus presumably easy for study subjects to use. A second group used PWS, the search-privacy tool that we designed and whose usability we wanted to test. The third group used TPTV; we chose TPTV, because it is regarded as the simplest and easiest-to-use of the official Tor distributions, and Tor [61] is the most widely used anonymity network. Our premise was that, if TPTV is the easiest general web-privacy tool for typical users to install and use, then a finding that PWS is easier to use than TPTV would be significant.

Users in the second and third groups had to install the assigned Firefox extension (PWS or TPTV, respectively). All users had to solve as many *search tasks* as they could in 45 minutes. A search task is a trivia question with an unambiguous, correct answer, together with a search method (Google, PWS, or TPTV). (See Appendix A for the list of trivia questions that users were given.) From the data collected, we were able to compare both PWS users and TPTV users with the control group with respect to accuracy, speed, and unrecoverable errors that they encountered. At the end of each session, we asked subjects to fill out a short survey about their privacy concerns and their reasons for adopting search-privacy tools (or choosing not to adopt them).

The results support our hypothesis that PWS perfoms better than TPTV. PWS users encountered fewer unrecoverable errors and were able to solve search tasks more effectively than TPTV users. Moreover, PWS users did not encounter as much of a performace degradation as expected (in comparison with Google users). However, the survey results indicate that users have very little tolerance for increased latency

in Web search – they do not want to wait longer than they are accustomed to for their search results. This lack of tolerance for delay will be a significant challenge in the design of search-privacy tools that can be widely adopted.

The reminder of this chapter is organized as follows. Section 3.2 describes related prior work. Sections 3.3 and 3.4 give brief overviews of PWS and TPTV, respectively. Section 3.5 presents the study design, and Section 3.6 presents the results. We draw conclusions in Section 3.7.

3.2 Related Work

Clark, van Oorschot, and Adams [8] presented a set of meaningful criteria with respect to which one should evaluate the usability of Tor-based Web-privacy technology; we used these criteria in the design of our study. Clark *et al.* also used a cognitive walkthrough to evaluate the usability of several Web-privacy tools. By contrast, we evaluate PWS and TPTV with a user study that yields empirical results.

Dingledine and Mathewson [13] studied the relationship between anonymity networks (such as Tor) and usability. They make the point that usability has a positive effect on the overall security of an anonymity network. This motivates our study of the usability of PWS and TPTV, because both are Tor-based.

By contrast, TrackMeNot [65] is a Firefox extension that seeks to enhance privacy by adding "cover traffic" to each user's query stream. In addition to sending the user's real queries to the search engine, a TrackMeNot-enabled browser also sends a random stream of "fake" queries; the claim is that this can be done in a such a way that the search engine cannot distinguish real queries from fake and hence cannot accurately profile the user. The cover-traffic approach to search privacy is incomparable to the Tor-based approach used by PWS; therefore, we did not include TrackMeNot in our

study.

3.3 Overview of PWS

In this section, we briefly review the design of PWS and the major difference between it an earlier web-privacy tools, including the TPTV bundle. Details can be found in Chapter 2.

PWS is comprised of several modules that collaborate to handle sensitive information on each "level" of the interaction of browser and search engine (see Figure 3.1). When the user executes a query using PWS, the browser connects to the local HTTP proxy. The proxy filters the HTTP request, then sends it to the search engine over the Tor network. Later, the proxy receives the response from Google through Tor, filters the HTML to remove all "active components," and sends the answer back to Firefox for display. Active components are programs (written in Javascript, Flash, Java, and many other popular languages) that run in Web pages; they are essential aspects of the functionality provided by many websites, and thus general web-privacy tools such as TPTV *cannot* remove them without destroying users' ability to use the Web. Unfortunately, active components can send large amounts of sensitive information (including personally identifying information) to a server from the client in whose browser they are running, and so they can destroy the privacy gained by routing the client-server interaction over Tor. A major contribution of the work in Chapter 2 was the observation that, while it is infeasible for a *general* web-privacy tool to remove active components from web pages and maintain functionality, it is feasible to remove them and maintain *search* functionality. This is precisely what PWS does.

Thus far, PWS can only be used with Google [29]; it would be straightforward

to extend it to let users select from multiple search engines.

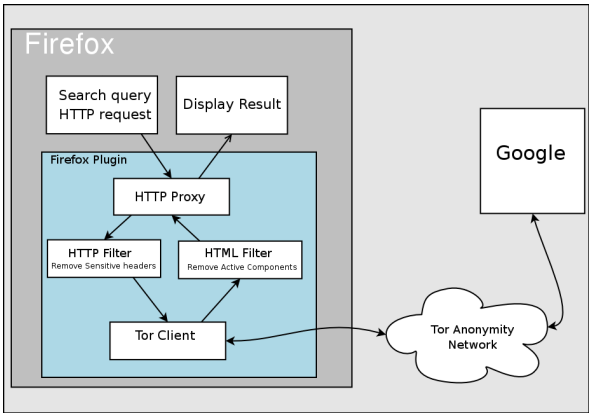


Figure 3.1: PWS Architecture

3.4 Overview of TPTV

The TPTV bundle is the standard Tor distribution for Windows users who want to navigate the Web anonymously. As its name suggests, TPTV is not a single program but rather a combination of applications each of which does part of the job:

Tor is a widely used anonymity network (more precisely, a network that defends users against traffic analysis [61, 14]).

Privoxy is a local Web proxy that controls the information "leaked" by browsers [48].

TorButton is a Firefox extension that manages and simplifies the configuration of Tor and Firefox and the interaction between them [63, 64]. In addition, TorButton prevents the browser from leaking several types of sensitive information while attempting to maintain functionality of most websites. In particular, it blocks known Javascript leaks, including window size, timezone, and user agent, but does not remove the Javascript from the displayed page; it disables caches that may be exploited by timing attacks; and it prevents Firefox from storing information on the

client machine in several ways that are known to cause leaks (*e.g.*, saving passwords and other forms of autocompletion).

Vidalia is the user interface for Tor. It allows the user to monitor and configure Tor in a user-friendly manner [69].

The Windows installer is used to install and configure all pieces of the TPTV bundle. We chose this setup for our study, because it requires almost no configuration and thus is likely to be the easiest way for most people to get started on using TPTV.

3.5 User-Study Design

Clark, van Oorschot, and Adams [8] identify the following *Core Tasks* that a user should be able to accomplish in a Tor-based system:

CT-1 Successfully install Tor and the components in question.

CT-2 Successfully configure the browser to work with Tor and the components.

CT-3 Confirm that the web traffic is being anonymised.

CT-4 Successfully disable Tor and return to a direct connection.

All of these core tasks informed the design of our study. We also included a fifth task that is specific to Web search: Successfully search the Web using the assigned Firefox extension (PWS or TPTV). To accomplish this, we presented users with a set of simple but realistic search tasks.

Clark *et al.* [8] also identify the following usability guidelines for Tor-based systems:

G1 Users should be aware of the steps they have to perform to complete a core task.

G2 Users should be able to determine how to perform these steps.

G3 Users should know when they have successfully completed a core task.

G4 Users should be able to recognize, diagnose, and recover from non-critical errors.

G5 Users should not make dangerous errors from which they cannot recover.

G6 Users should be comfortable with the terminology used in any interface dialogues or documentation.

G7 Users should be sufficiently comfortable with the interface to continue using it.

G8 Users should be aware of the application status at all times.

In addition to G1, G2, and G3, which refer to the user's ability to perform Core Tasks, we consider G4 and G5 particularly important. This is because users' encountering errors from which they can't recover seems like an obvious barrier to adoption; furthermore, it is something that we can measure in our study.

Preparation. Before beginning our experiment, we submitted our study design to Yale's human-subjects-research authorities for Institutional Review Board (IRB) approval. Our study was promptly classified as "minimal risk" and qualified for expedited approval. The only problem we encountered concerned remuneration of users. We originally proposed to pay users $20 per session, but we were directed to reduce it to Yale's standard $10 per session. The fact that $20 per session would not have blown our budget was considered irrelevant by the people in the IRB office; they did not want us to destabilize the campus-wide market for experimental subjects.

Sessions. Sessions were divided into three parts. In the first, we measured users' ability to complete installation and setup successfully; control-group users did not have to do this part. In the second, users completed as many search tasks as they could in 45 minutes. The third part was devoted to the survey described in Subsection 3.5.4 below.

Sessions took place in a medium-sized office that accommodated up to three subjects at a time, together with the experiment monitor. (The first author of this chapter served as experiment monitor for all of the sessions, but anyone familiar with all of the software involved could easily have performed this duty.) The office used for these sessions was not used by anyone else for any purpose throughout the duration of the study.

Subject Recruiting. Users were recruited from the population of Yale University students who use computers and Google daily. Recruiting was accomplished by posting signs in popular spots that offered a $10 payment for participation in a session.

Data Collection. Users interacted with a Web data-collection application that give them search tasks and received their answers; this allowed us to determine how many of the trivia questions each user answered correctly. The Web data-collection application also enforced the 45-minute time limit. Network traffic was monitored on the laptops that subjects used by means of a non-intrusive Firefox extension, and each URL visited as a result of each query issued was recorded; this allowed us to determine whether the subject actually found the answer to the trivia question on the web or already knew it before he or she started searching. It also allowed us to determine whether the subject actually used the requested search method.

In addition to the automatic data collection, the experiment monitor kept track of every time the subject asked for help in restarting the experiment after an unrecoverable error.

Starting point. Users were assigned a laptop computer with standard specifications (Win XP, Firefox 2.0). The initial configuration was saved as a virtual machine in order to ensure that every user started from exactly the same configuration.

User Training. Once users were familiar with the environment, they were given brief instructions to ensure that all users could find the Web data-collection application when they needed it.

Search Tasks. The core of the experiment was a set of search tasks, each of which was defined by a trivia question and a search method (Google, PWS, or TPTV). As a response to a search task, the user was required to give an answer to the question and the resource (URL) at which the answer was found. In order to be able to evaluate users' success objectively, the answers to the questions in search tasks had to be verifiable facts. To ensure that this would be the case, we used a commercially available trivia database as our source of questions [66]. Trivia questions are well suited to this experiment, because they have unambiguous, verifiable answers, and some of them seem to be hard enough to require more than one query. Moreover, the use of objective trivia questions allowed us to minimize the amount of personal information about the subjects that would be revealed in the search process: Trivia questions are broad and general; neither the search queries nor the answers are likely to contain private information about users.

In the form in which one downloads them from [66], the database questions are grouped by topic and sorted in increasing order of difficulty within topic. In order to avoid bias that could result from users' widely varying familiarity with each topic and to provide users with questions of all levels of difficulty, the database questions were randomly permuted before being given to the study subjects. Only one permutation was performed, however; all study subjects received the questions in the same order.

3.5.1 Installation

To get started, users in the PWS and TPTV groups were asked to install the assigned software. We pointed them to the relevant download site and corresponding instruc-

tions. We verified whether the installation was done correctly and helped them to finish correctly if they had failed to do so on their own.

3.5.2 Switching

The goal of this step was to test whether users in the PWS and TPTV groups could successfully switch between their assigned privacy tool and Google. (Once again, users in the control group skipped this step.) In this step, the content of the questions was not relevant. Users were assigned a set of easy search tasks some of which were supposed to be performed with the assigned privacy tool and some with plain Google. Success was measured by counting the number of queries performed in the correct mode.

3.5.3 Searching

The goal of this step was to measure the performance penalty attributable to the use of PWS or TPTV. Performance was measured in terms of both speed and accuracy. Users in all three groups were given the same set of trivia questions and instructed to answer as many as possible, as accurately as possible, in the 45 minutes allotted. Both the total number of questions answered (*i.e.*, the total number of search tasks completed) and the total number of correct answers found on the web were recorded.

3.5.4 Survey

The final thing we did was to survey the subjects about their level of concern with web privacy and their reasons for using or not using brower-based privacy tools. The questions asked were:

SQ1A: When using Google to search the Web, do you avoid certain topics (select all relevant answers):

a) I never do.

b) At public places

c) On a computer shared with other people

d) At my workplace

e) At home

SQ1B: I avoid certain topics when using Google because (select all relevant answers):

a) I never do.

b) I'm concerned that other people with access to the computer will have access to my search history.

c) I'm concerned that someone may intercept the traffic between me and Google.

d) I'm concerned that Google will learn certain things about me.

e) My employer has a corporate policy governing personal use of company-owned resources.

f) Other:

SQ1C: If you never refrain from searching (*i.e.*, if you selected (a) in the previous question), why is this?

a) I don't consider my search history to be private data.

b) I do consider my search history to be private data, but I trust that it is well protected.

c) Other:

SQ2A: How much do you agree with the following statement: "When I use Google to search the Web, Google has a good chance of associating my identity with each of my queries if it chooses to do so."

a) Strongly disagree

b) Weakly disagree

c) Weakly agree

d) Strongly agree

SQ2B: How much do you agree with the following statement: "Google keeps a fairly complete search history associated with my identity."

a) Strongly disagree

b) Weakly disagree

c) Weakly agree

d) Strongly agree

SQ3: Suppose that Google were able to associate each query you issue with you, and you had an equally accurate alternative method for searching that protected your identity but performed more slowly. You would consider using it if getting the answer to a query (underline one answer per row):

a) Took an additional 1 second or less: Never, Sometimes, Always

b) Took about 5 additional seconds: Never, Sometimes, Always

c) Took about 10 additional seconds: Never, Sometimes, Always

d) Took about 30 additional seconds: Never, Sometimes, Always

e) Took about 60 additional seconds: Never, Sometimes, Always

SQ4: If Google were able to associate each query you issue with you, and you had an equally accurate alternative method for searching that protected your identity, you would consider it using it for queries about (select all relevant answers):

a) Health

b) Sex

c) Politics

d) Illegal activities

e) Yourself

f) People you know

g) Your job or your employer

h) Would never use it

3.6 Results

3.6.1 Installation

Google users (the control group) did not have to install any software, and hence there are no results to report on their success with installation. PWS and TPTV users

Search Method	Installation Success Rate
TPTV	59.9%
PWS	84.6%

Table 3.1: Percentage of users who successfully completed installation

were told to follow the installation instructions on the respective websites; when a user said that he or she was finished with this step, we checked whether the Firefox extension had been installed successfully. The results appear in Table 3.1. Because PWS uses the standard installation procedure for Firefox extensions, the significantly higher success rate for PWS users may be attributed to Firefox's high usability. It is worth noting, however, that we designed PWS so that Firefox users would have a single extension-distribution package precisely in order to make the setup process manageable for most users.

Of the 15.4% of PWS users who did not complete the installation process successfully, most failed by not confirming the installation of an untrusted extension. TPTV users experienced a more diverse set of failure modes. Although the instructions pointed to a single Windows package, some users downloaded the wrong package. Others failed to restart the browser after the installation process was done.

3.6.2 Switching

Users in the second and third groups were instructed to perform each of the first 5 search tasks using a randomly assigned search method – either plain Google or their assigned extension. A user "succeeded" in this experiment if he or she performed 4 of the 5 tasks using the correct search method. Results are presented in Table 3.2.

Switching performance for the two extensions was not as similar as these numbers suggest. 46.6% of the users in the TPTV group, when instructed to use TPTV for the first search task, failed to activate Tor. They knew what they were supposed

Search Method	Switching Success Rate
TPTV	73%
PWS	77%

Table 3.2: Percentage of users who successfully performed at least 4 out of 5 potential switches.

Percentage of Correct Answers in Completed Search Tasks		
Search Method	Average	Median
Google	92.1%	95.5 %
TPTV	86.3%	93.3 %
PWS	81.5%	92.0 %

Table 3.3

to do but could not find the user-interface element that they needed and asked the experiment monitor for help. Once they received that help, many of them were able to perform subsequent switches correctly. The large number of users who needed help activating Tor in the first task is reflected in Table 3.5 below.

3.6.3 Accuracy and Speed in Searching

Having chosen search tasks that are realistic and have factual, verifiable answers, we set out to measure the performance degradation that PWS and TPTV users suffered in terms of accuracy and speed. The results appear in Tables 3.3 and 3.4.

As can be seen in Table 3.3, neither PWS nor TPTV imposed a very large penalty on users with respect to accuracy, but both imposed some penalty. Given that

Number of Search Tasks Completed in 45 minutes		
Search Method	Average	Median
Google	35	36
TPTV	18.2	18
PWS	30.9	27

Table 3.4

TPTV provides users with exactly the same web-search interface as Google, why should TPTV impose any accuracy penalty at all? Plausible explanations include unrecoverable errors (as explained below) and the Tor-induced slowdown of the search process (which may have caused some users to give an answer based on the results of one query when several queries were required to find the correct answer).

It is not clear exactly why PWS imposes a higher accuracy penalty than TPTV, but the following observations seem relevant. The PWS user interface is different from the Google-TPTV interface, and it is likely that users are not familiar enough with it to search with peak accuracy; this would not be an insurmountable barrier to adoption, because they would become familiar with it over time if they continued to use it, and the initial accuracy penalty is not prohibitive. Some of Google's helpful features, like query suggestion and spell checking, are not present in this early version of PWS, and their absence probably hurts accuracy. It is also worth mentioning that PWS only provides the top 20 results in a single page.

In terms of speed, measured by how many search tasks users could answer in a fixed amount of time, Google set the benchmark at an average of 35 tasks in 45 minutes. For detailed results, see Table 3.4. The poor performance of the TPTV group can be explained by two basic facts. First, TPTV uses Tor for all Web requests, and Tor increases latency considerably; PWS is faster in part because it uses Tor only for the interaction with Google. This difference is a natural consequence of different adversary models and is explained in Chapter 2. Second, TPTV users encountered several problems that slowed them down. The relevance of a user's encountering this type of frustrating problem during his or her first hour of experience with a search-privacy tool goes beyond accuracy; it can have a direct and severely negative impact on adoption.

Search Method	Percentage of Users who Made No Unrecoverable Errors
Google	100.0%
TPTV	20.0%
PWS	76.9 %

Table 3.5

3.6.4 Unrecoverable Errors in Searching

As discussed in Subsection 3.6.1 above, some users were unable to complete the installation process and needed help before they could proceed with the rest of the session. After installation, some users encountered other errors from which they could not recover on their own. The figures in Table 3.5 include both classes of unrecoverable error.

It is interesting to consider the type of unrecoverable errors that TPTV users encountered. Besides not being able to install the software correctly, they faced three significant problems.

33.0% were faced with a Google page in a language they cold not understand.

33.0% were told by Google that it could not answer queries because the user's machine was infected by spyware.

46.6% were not able to figure out how to activate TPTV after installing.

As a result, a significant fraction (80%) of TPTV users could not finish the study without help from the experiment monitor.

3.6.5 Survey

We now report the users' answers to the survey about their privacy concerns and their willingness (or the lack thereof) to use certain types of privacy technology.

SQ1A: When using Google to search the Web, do you avoid certain topics (select all relevant answers):

21.95% I never do.

60.98% At public places

68.29% On a computer shared with other people

56.10% At my workplace

14.63% At home

SQ1B: I avoid certain topics when using Google because (select all relevant answers):

17.07% I never do.

65.85% I'm concerned that other people with access to the computer will have access to my search history.

19.51% I'm concerned that someone may intercept the traffic between me and Google.

26.83% I'm concerned that Google will learn certain things about me.

34.15% My employer has a corporate policy governing personal use of company-owned resources.

00.00% Other

SQ1C: If you never refrain from searching (*i.e.*, if you selected (a) in the previous question), why is this?

50.00% I don't consider my search history to be private data.

50.00% I do consider my search history private data, but I trust it is well protected.

00.00% Other

SQ2A: How much do you agree with the following statement: "When I use Google to search the Web, Google has a good chance of associating my identity with each of my queries if it chooses to do so."

12.20% Strongly disagree

29.27% Weakly disagree

36.59% Weakly agree

21.95% Strongly agree

SQ2B: How much do you agree with the following statement: "Google keeps a fairly complete search history associated with my identity."

7.3% Strongly disagree

14.6% Weakly disagree

53.6% Weakly agree

21.9% Strongly agree

The answers to SQ3 are given in Table 3.6 and Figure 3.2. It is quite interesting that some of these results, especially those in Figure 3.2, indicate that users who experienced more privacy-related delay when trying to complete the search tasks expressed less willingness to trade increased latency for increased privacy. In general, control group users, who searched with plain Google and thus experienced no privacy-related delay, expressed much more willingness to trade latency for identify protection

Seconds	Aggregate			Google		
	Always	Sometimes	Never	Always	Sometimes	Never
at most 1	97.56	2.44	0.00	100.00	0.00	0.00
about 5	90.24	7.32	2.44	100.00	0.00	0.00
about 10	56.10	34.15	9.76	69.23	23.08	7.69
about 30	17.07	36.59	46.34	46.15	30.77	23.08
about 60	7.32	34.15	58.54	23.08	38.46	38.46
Seconds	PWS			TPTV		
	Always	Sometimes	Never	Always	Sometimes	Never
at most 1	100.00	0.00	0.00	93.33	6.67	0.00
about 5	92.31	7.69	0.00	80.00	13.33	6.67
about 10	53.85	38.46	7.69	46.67	40.00	13.33
about 30	7.69	38.46	53.85	0.00	40.00	60.00
about 60	0.00	38.46	61.54	0.00	26.67	73.33

Table 3.6: Percentage of users who would trade N seconds of delay for identity protection, aggregated and by group, for $N \in \{1, 5, 10, 30, 60\}$

than users in groups 2 (PWS) and 3 (TPTV). TPTV users, who experienced the most delay, were also the most likely to say that they would *never* trade N seconds of delay for identity protection, for all values of $N \in \{5, 10, 30, 60\}$.

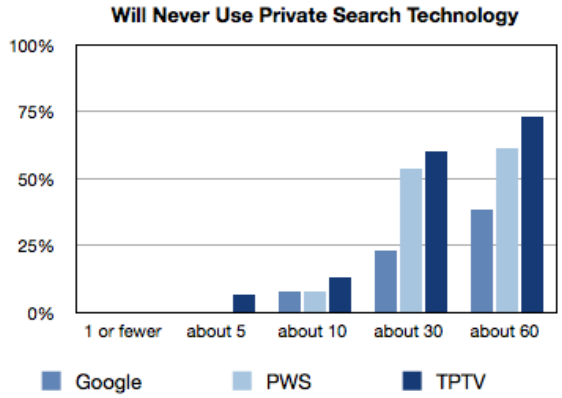


Figure 3.2: Percentage of users in each group who said they would **never** trade N seconds of delay for identity protection, for $N \in \{1, 5, 10, 30, 60\}$

Study subjects were given no tangible incentive to excel in their assigned tasks. Nonetheless, most of them strove hard to perform the search tasks as quickly as possible. Anecdotally, we can report that several PWS and TPTV users found their

experiences quite frustrating and explicitly expressed anger (even fury) by the end of the session. According to them, this frustration was caused by delay: The Firefox extensions they were using did not allow them to complete search tasks as fast as they wanted to.

SQ4: If Google were able to associate each query you issue with you, and you had an equally accurate alternative method for searching that protected your identity, you would consider it using it for queries about (select all relevant answers):

75.61% Health related queries

92.68% Sexual related content

53.66% Political related queries

87.80% Illegal activities

21.95% Things about yourself

7.32% Things about people you know

12.20% Job related queries

3.7 Conclusions

From one point of view, the results of this study make PWS look good. At the expense of a small degradation in accuracy, which might disappear over time as they gained experience with it, non-expert users found PWS to be easier to install and faster to use for search than TPTV. This is evidence that it is good to provide users with a tool aimed specifically at *search* privacy: Such a tool can be made both more secure (because pages can be stripped of active components) and more usable than a general web-privacy tool.

From another point of view, the results cast doubt on the worth of this well established approach to search privacy: Users said that delays caused by the Tor network make any Tor-based system, including both PWS and TPTV, highly undesirable for search. One rule of thumb in the study of Web applications is that delays of more than 10 seconds cause users to lose focus [45]. In our study, the average time to completion for a search query was approximately 30 seconds, for both PWS and TPTV. (Note that the fact that it was 30 for both does not contradict the results in Table 3.4, because many search tasks require multiple queries.) Despite the fact that Tor is the best widely available anonymity network, it is too slow for use in search. Tor designers and coders are well aware that latency is a barrier to building usable Web applications on top of Tor [15] and are working to improve the situation. However, because design, implementation, and deployment of a very low-latency anonymity network is a difficult open problem, it is unrealistic to expect improvement in the near future.

Finally, we note some obvious limitations of our study design. Searching for answers to trivia questions is only one special case of Web search. It is possible that privacy technology would affect users' overall search activity in ways not capturable in a study of this special case. For example, Google enhances the results it displays to a user based on that user's long-term search history, but that history is likely to be irrelevant to searches for the answers to the questions in Appendix A and was in any case unavailable to Google in this study. Despite this limitation, our comparison of three well-defined groups' performance on trivia questions does shed some light on the effects of privacy technology in this special case. Another limitation of our study concerns the definition of "unrecoverable error." We considered an error "unrecoverable" if the subject gave up and asked the experiment monitor for help; it is possible that, without the implicit pressure imposed by the fact that this was an

experimental session of limited duration, a determined user who had as much time as he or she was willing to devote to it could "recover" from one or more of these errors.

Chapter 4

Threshold Privacy and Its Application to Distributed Alert Sharing

4.1 Introduction

The need to share information while maintaining participants' privacy arises in many application scenarios. This has, in recent years, led to extensive work on privacy-preserving data mining [36]. Sharing network-security information is one such application scenario. With widespread penetration of malicious software such as computer worms, it has become more important for the operators of local networks to share information about intrusions and other unwanted activities in their networks so that a collective view is available for detecting and preventing such activity.

In fact, there are several sites, e.g., DShield.org [67] and CyberTA.org [59], that collect and share network alerts generated locally by anomaly-detection systems such as Snort [58] and Bro [47]. When the alerts are pooled, threats that concern *multiple*

local networks can be distinguished from traffic that is deemed "anomalous" at *one particular local network*.

However, even with these sites, current information sharing is very limited. The main hurdle lies in the fact that, in order to make the sharing useful for security purpose, it is necessary to share *details* of the "suspicious" activity and, at the same time, protect the privacy of the data contributor. (The collector site is not trusted to protect privacy.) Also, the sharing framework must be scalable to accommodate many participants. This sharing problem can be modeled as *privacy-preserving threshold-union*. If $A_1, \ldots, A_n$ are sets of alerts generated at n sites, their *t-threshold union* A consists of all alerts a such that a occurs in at least t distinct sets $A_{j_1}, \ldots, A_{j_t}$. Kissner and Song [33] put forth the thesis that, for sufficiently large t, elements of the t-threshold union would not be considered privacy sensitive by network owners, because the very fact of their having been detected by at least t networks means that they cannot depend intimately on the proprietary state of any one network[3].

Existing protocols for computing threshold union are not sufficiently scalable for practical purposes. For example, Kissner and Song's protocol [33] has communication complexity $\Omega(n^2 \cdot c \cdot \log P)$, where n is the number of participating networks, c is the maximum, over $1 \leq j \leq n$, of the number of distinct alerts in A_j, and P is the size of the input domain (*i.e.*, the total number of distinct alerts a that may be generated). More generally, privacy-preserving threshold-union computation is an example of *secure function evaluation*, for which an elaborate theory has been

[3]Although we adopt Kissner and Song's premise that t or more sites that experience the same security alert can reveal that fact without compromising their privacy, we note that there are reasons to question it. For example, if an attacker devises a way to trigger the alert system whenever some user on the local network is viewing pornography, network operators may prefer not to share those alerts, regardless of whether at least $t - 1$ other networks experienced them. Despite potential objections of this sort, Kissner and Song's claim that t-threshold union is applicable to network monitoring is widely believed, and we think that it is sufficiently plausible to justify further study of t-threshold-union protocols.

developed since Yao's original paper of more than a quarter-century ago [71], but no secure-function-evaluation protocol in the literature achieves the scalability that we seek. Furthermore, these protocols require that every contributor be available *at the same time* when computing the union. This is very different from current practice of sites such as DShield.org and CyberTA.org, in which the contributors can upload information at any time.

The main contribution of this chapter is a protocol for t-threshold union. The protocol is scalable, because it has linear communication complexity and adopts the centralized architecture used by most sharing sites. The tradeoff is that our privacy guarantee is weaker than the one offered by secure-function-evaluation-based protocols. It is based on the *entropic security* property developed by Dodis and Smith [16]; roughly speaking, entropic security is similar to semantic security but can only be guaranteed if the plaintext distribution has high entropy.

One ingredient of our solution is *threshold cryptography* [11, 10]. Although there is a well developed theory of threshold cryptography that features protocols based on such standard constructions as RSA signatures and encryption [4] and discrete-logarithm key generation [22], we cannot use it directly. In standard threshold-cryptography schemes, one party encrypts the plaintext, and then multiple parties, each holding a share of the decryption key, compute partial decryptions that they can pool to recover the plaintext if and only if together they have at least t distinct decryption-key shares, where t is the threshold. In our canonical application of threshold union (security-alert sharing), no one party possesses all of the relevant private-key fragments, and thus the encryption process must be distributed across all of the participating sites.

In our protocol, each participating site j uses a threshold encryption function to encrypt plaintexts, each of which consists of an alert $a \in A_j$ and a hash value $\mathsf{H}(a)$.

The encryption is done in such a way that one bit of the hash is revealed and, once the threshold t is met for that bit, the next bit is revealed. Inputs that meet the threshold will be completely revealed. For inputs that do not meet the threshold, some bits of the hash will be revealed.

The hash values enable the protocol to group the ciphertexts that were generated from the same alert, but they may also reveal some information about the plaintext. However, we prove in Section 4.4 below that, in the random-oracle model, if privacy-sensitive plaintexts have high min-entropy, the overall protocol is entropically secure.

The rest of the chapter is organized as follows. In Section 4.2, we give the statement of the privacy-preserving threshold-union problem and explain our protocol requirements in more detail. In Section 4.3, we give a detailed description of how alerts are encrypted and later decrypted to complete the specification of the protocol. Efficiency and security are discussed in Sections 4.4 and 4.5, respectively. Section 4.6 gives an idea of what to expect from a real application by presenting results of simulations based on real network data. We discuss malicious contributors in Section 4.7.

4.2 Participants, Requirements, and Protocol Structure

We illustrate an alert-sharing system that uses our protocol in Figure 4.1. It consists of the following components:

Contributors: These are the sites $\mathcal{S}_1, \ldots, \mathcal{S}_n$ at which alert data are collected. Let A_j be the set of alerts collected by contributor $\mathcal{S}_j$.

Collector: This is the central party $\mathcal{C}$ to whom contributors send encrypted alerts and corresponding hash values. $\mathcal{C}$ computes the threshold union and distributes

the results to the contributors (and perhaps to others, in the most extreme case by posting them on a public website).

Registration Authority: This is the central party $\mathcal{RA}$ that generates and distributes encryption keys to the contributors. One could replace $\mathcal{RA}$ by a distributed key-generation protocol as in [22], but we do not consider that generalization further in this chapter .

Note that $\mathcal{RA}$ and $\mathcal{C}$ are distinct and play different roles in the protocol. The contributors interact with $\mathcal{RA}$ just once when they join the system. Subsequently, they interact with $\mathcal{C}$ periodically. During each operational period, contributors run their alert detectors, encrypt the alerts, and send the results to $\mathcal{C}$, whom they trust to perform the t-threshold-union computation and distribute the results correctly. Note that contributors need *not* trust $\mathcal{C}$ to safeguard their sensitive inputs: The consensus-privacy property ensures that inputs that are not in the computed threshold union, *i.e.*, those that are truly private, are not revealed in the course of the computation. The protocol might be performed once a day, once an hour, or even more frequently, depending on the severity of the threat environment and the volume of data involved; the volume will be determined by n (the number of contributors), c (the maximum number of alerts generated by any contributor), and P (the total number of distinct, possible alerts, which affects the number of bits required to communicate the encryption and hash of each one).

The beginning of each operational period is a completely fresh start. That is, in period w, the collector $\mathcal{C}$ should be able to decrypt a particular alert a if and only if a is contributed by at least t sites *during* w. $\mathcal{C}$ should not be able to decrypt a by using shares contributed both in period w and in previous periods. In order to ensure this, the contributors are required to include both the alert and the period number in the plaintext that they encrypt and hash. In this way, an alert a in period

u will be considered different from alert a in period v.

Our threshold-union protocol should satisfy the following criteria:

Accommodates transient contributors: We encourage registration of a large, diverse set of contributors not all of whom will participate in every period. Interactions should be short-lived, and the protocol should terminate with meaningful results even if one or more contributors drops out in the middle of a period.

Preserves privacy of sensitive inputs: No contributor wants to share data that may reveal problems unique to its site; as discussed in Section 1, the consensus-privacy property captures this criterion.

Scalability: The system should perform well when thousands of contributors send hundreds of inputs in each period, even when periods are short. This motivates the use of a central collector rather than a multi-round, distributed protocol. (Recall that the communication complexity of Kissner and Song's protocol [33] grows quadratically with the number of contributors; this is in fact true of all known solutions based on secure function evaluation.)

Note that we are *not* trying to achieve contributor anonymity in this protocol. There are two main reasons for this. First, the basic premise of the threshold-union approach is that any alert that is sent by t or more contributors is, by definition, not private; the individual contributor isn't revealing anything about his own network by revealing the fact that he was a victim of this indiscriminate attack. Second, contributors aren't trying to hide from the collector or from others the fact that they sent something to the collector in a particular operational period; this is a cooperative system in which all participants are doing something that they should be commended for – sharing information in order to improve Internet security. The only things that they want to hide are the alerts that *are* specific to their own sites, and those are the ones that won't be decrypted, because the number of shares

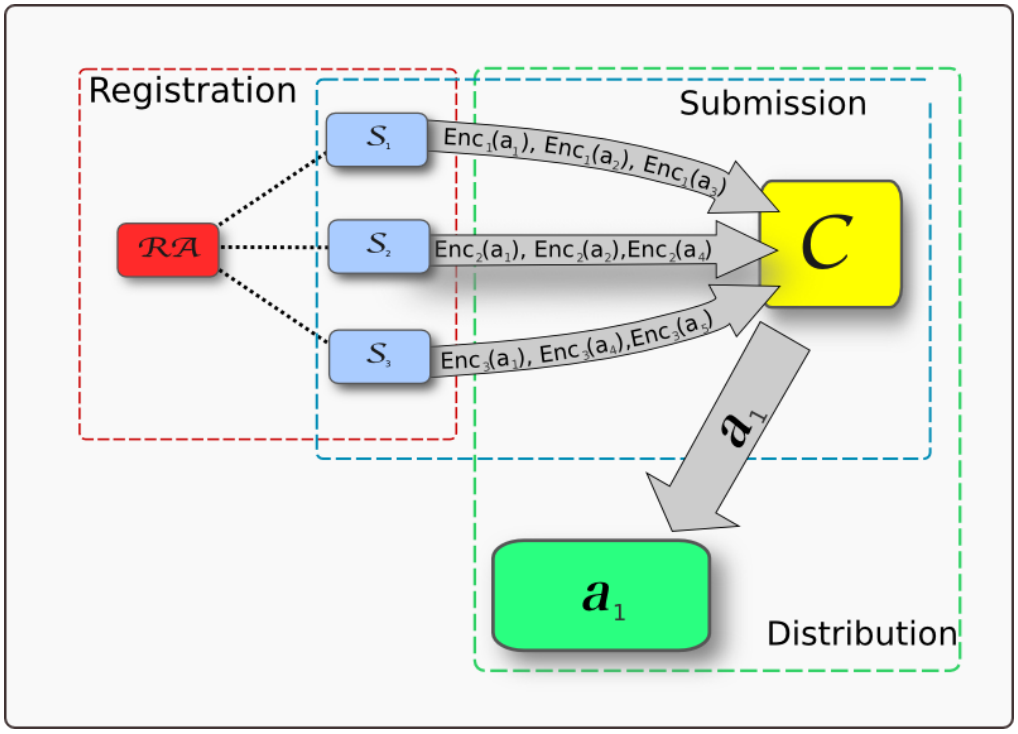


Figure 4.1: System architecture: $\mathcal{RA}$ and $\mathcal{S}_i$ interact only once in the registration stage. During the submission stage, the contributors $\mathcal{S}_1, \mathcal{S}_2$ and $\mathcal{S}_3$ send the encryptions of their inputs to $\mathcal{C}$. In the last stage, $\mathcal{C}$ decrypts and publishes the elements that have been sent by at least t contributors. In this example, only input a_1 is published for threshold $t = 3$.

submitted is insufficient. If a participant disrupts the system by spamming it with an excessive number of contributions in a given period, then he should be identified and blocked.

4.3 Messages

In the previous section, we defined the basic structure of the protocol. We will now specify how the messages that participants send to the Collector are encrypted and how the collector can combine them to recover the original alerts when the threshold condition is met.

A starting point is to instantiate a threshold-encryption scheme. Using his share of the private key, each contributor can then generate a share of an alert and submit it to the Collector. If the Collector receives a number of shares that is greater than or equal to the threshold for a particular alert, he can decrypt it. The Collector is trusted to publish the results.

A Threshold Public Key Encryption (TPKE) scheme is defined by the following set of functions. Here, s is the security parameter, t is the threshold, and n is the number of contributors.

•$KeyGen(1^s)$: The key-generation algorithm produces a public key Pub and private-key shares Pri_i for $\mathcal{S}_i$, $1 \leq i \leq n$.

•$Encrypt(a, \mathsf{Pub})$: Given a plaintext message a and a public key Pub, the encryption function produces the ciphertext C.

•$PartialDecrypt(\mathsf{C}, \mathsf{Pri}_i)$: Given a ciphertext C and a share Pri_i of the private key, the partial-decryption function produces the corresponding share C_i.

•$Combine(\{\mathsf{C}_i\}_{i\in\mathsf{T}})$: Given a set $\{\mathsf{C}_i\}_{i\in\mathsf{T}}$ of shares, the combining function produces the plaintext message a if $|\mathsf{T}| \geq t$ and otherwise produces no output at all.

In our alert-sharing scenario, it is natural to consider the following composition of two threshold-crypto building blocks:

•$Share(a, \mathsf{Pri}_i, \mathsf{Pub}) = PartialDecrypt(Encrypt(a, \mathsf{Pub}), \mathsf{Pri}_i)$: Given a plaintext message and a private-key share Pri_i, the share function produces $\mathcal{S}_i$'s input to the combining function.

Even though using $Share(a, \mathsf{Pri}_i)$ as a submission by contributor $\mathcal{S}_i$ for each alert a seems like a good idea at first, it presents two problems that need attention. First, there is no efficient way for the Collector to identify sets of shares that correspond to the same alert a in order to decrypt them. If the Collector had unlimited time and/or computational power, he could try all subsets of size t of the inputs to find subsets that decrypt correctly. Given realistic amounts of time and computational power, of course, this exhaustive-search approach is infeasible.

The second problem lies in the encryption process itself. Clearly, semantic security [28] would be desirable. However, if an encryption scheme is semantically secure, then the computation of $Encrypt(a, \mathsf{Pub})$ must be probabilistic. This is true, for example, of threshold ElGamal [24]. In our application, this would require all contributors to use the same random bits. The problem lies in the fact that threshold cryptosystems are designed to have one party encrypt the plaintext and then distribute shares of the ciphertext. We need to have multiple contributors perform the encryption.

A viable alternative would be to use a deterministic threshold scheme, like threshold RSA [11], but that would introduce the well known security problems inherent in deterministic public-key cryptography. One could envision solving this problem with some secret known to all but the adversary. This, however, would make the protocol insecure in scenarios in which one of the contributors that knows the secret colludes with the adversary.

We will present an alert-sharing scheme that solves both of these problems. Our main ingredient is Threshold Identity Based Encryption (TIBE), a threshold variant of Identity Based Encryption. Using TIBE, we can generate semantically secure threshold encryptions without interaction. To solve the problem of "grouping" messages that belong to the same alert, each message will reveal a small number of bits of the hash of the alert that gave rise to it. These bits allow the collector to match the corresponding messages to achieve efficient decryption. Although revealing some bits of the hash of the alert does preclude semantic security, if the number of revealed bits is small in comparison to the expected size of the message space, then a reasonable form of security can still be proven.

4.3.1 Threshold Identity Based Encryption

Identity Based Encryption (IBE) is a public-key cryptography variant in which the public key can be an arbitrary bit string. For example, it is possible to encrypt an e-mail message using the recipient's email address as the public key (even before the recipient generates his private key!) without having the recipient's certificate. The drawback is that the IBE infrastructure involves a Private-Key Generator (PKG) that can generate everybody's private keys. Note that such a PKG could decrypt any message in the system.

Threshold Identity Based Encryption, a natural extension of IBE, distributes the trust in the PKG among a set of servers in a threshold manner. In a TIBE scheme, none of the servers can generate a private key on its own, but each can generate a share of a private key. When enough shares are combined, the actual private key can be reconstructed. This precludes the possibility that a centralized PKG could become corrupt and misuse private keys.

In [3], the authors construct a Threshold Public Key scheme on top of a TIBE

scheme. In a similar way, we build our alert encryption scheme using a TIBE scheme as foundation.

A TIBE scheme, as defined in [3], consists of the following functions.

•$Setup_{TIBE}(n, t, 1^s)$ takes as input the number n of decryption servers, a threshold t, where $1 \leq t \leq n$, and a security parameter $s \in Z$. It outputs a triple $(\mathsf{Pub}, VK, \mathsf{Pri})$, where Pub is *the system parameter*, VK is the *verification key*, and $\mathsf{Pri} = (\mathsf{Pri}_1, ..., \mathsf{Pri}_n)$ is a vector of master-key shares analogous to the private-key shares in the definition of TPKE. Decryption server i is given the master-key share (i, Pri_i).

•$ShareKeyGen_{TIBE}(\mathsf{Pub}, i, \mathsf{Pri}_i, ID)$ takes as input the system parameter Pub, an identity ID, and a master-key share (i, Pri_i). It outputs a private-key share $\theta = (i, \hat{\theta})$ for ID.

•$ShareVerify_{TIBE}(\mathsf{Pub}, VK, ID, \theta)$ takes as input the system parameter Pub, the verification key VK, an identity ID, and a private key share θ. It outputs `valid` or `invalid`.

•$Combine_{TIBE}(\mathsf{Pub}, VK, ID, \theta_1, ..., \theta_t)$ takes as input Pub, VK, an identity ID, and t private-key shares $\theta_1, ..., \theta_t$. It outputs either a private key d_{ID} or $\perp$.

•$Encrypt_{TIBE}(\mathsf{Pub}, ID, a)$ takes Pub, an identity ID, and a plaintext message a and outputs a ciphertext C.

•$ValidateCT_{TIBE}(\mathsf{Pub}, ID, C)$ takes as input Pub, an identity ID, and a ciphertext C. It outputs `valid` or `invalid`. If `valid`, we say that C is *a valid encryption under* ID.

•$Decrypt_{TIBE}(\mathsf{Pub}, ID, d_{ID}, C)$ takes as input Pub, ID, a private key d_{ID}, and a ciphertext C. It outputs a message M or $\perp$.

4.3.2 Bit-by-bit decryption

We will now show how, using a TIBE scheme, each contributor can encrypt alerts to be sent to the collector. We will also show how the collector can completely reveal messages submitted by more than t contributors. We assume that TIBE keys have been generated using $Setup_{TIBE}$ and distributed to the appropriate parties. In particular, every contributor i has a private key Pri_i, and the public key Pub is known to all.

Recall that execution is divided into operational periods. Let w be the operational-period index, $\mathsf{H} : S_a \rightarrow \{0,1\}^k$ be a hash function (like sha-256), and $pre_l(x) : \{0,1\}^k \rightarrow \mathbb{N}$ the l-bit prefix of string bit x.

To submit an alert a, contributor i uses the following nested encryption algorithm to generate a share.

$Share_{TIBE}(a, \mathsf{Pri}_i, i)$:

1: $m \leftarrow a$
2: **for** $p = k$ down to 1 **do**
3: $\quad id \leftarrow (w, pre_p(\mathsf{H}(a)))$
4: $\quad \theta \leftarrow ShareKeyGen_{TIBE}(\mathsf{Pub}, i, \mathsf{Pri}_i, id)$
5: $\quad m \leftarrow (id, Encrypt_{TIBE}(\mathsf{Pub}, id, m), \theta)$
6: **end for**
7: **return** m

$Share_{TIBE}()$ operates by encrypting the message multiple times in layers. When it is decrypted, each layer will reveal an additional bit of $\mathsf{H}(a)$.

In line 3, the identity id for the encryption layer is computed. That identity contains the period identifier and a prefix of the hash. In each iteration, one fewer

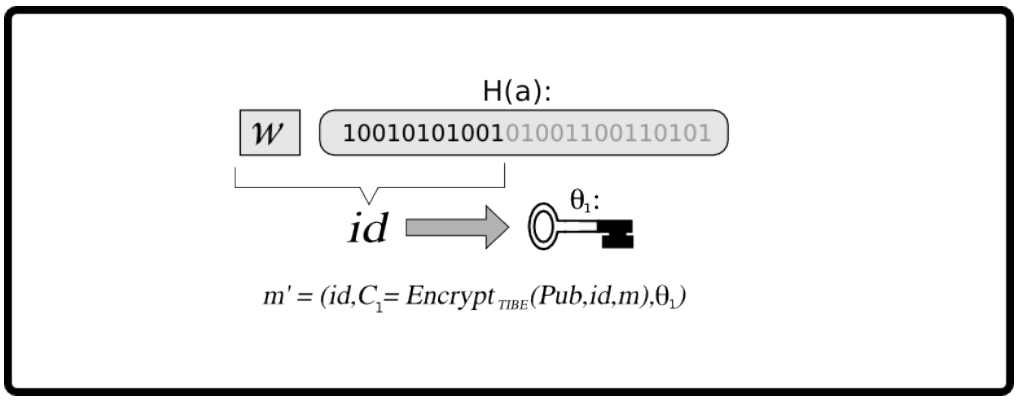


Figure 4.2: Illustration of a single iteration of $Share_{TIBE}(a, \mathsf{Pri}_1, 1)$ for participant $\mathcal{S}_1$. In each iteration, a prefix of the hash $\mathsf{H}(a)$ is used as part of the identity id. Using that id, $\mathcal{S}_1$ can generate a share of the private key θ_1 corresponding to identity id.

hash bit is included in id. Line 4 generates a $TIBE$ share of the private key corresponding to id. Finally, in line 5, the result of the previous iteration is encrypted with the public encryption function $Encrypt_{TIBE}()$. The result of line 5 is a triple (id, C, θ), in which θ is a share of the private key corresponding to the identity id in the underlying $TIBE$ scheme. When t shares of the key are combined, the full public key corresponding to id will be recovered, enabling decryption of any message encrypted using id as a public key.

The Collector receives triples of the form $(id, C, share_i)$ from the contributors. At the the top level, id reveals one bit of $\mathsf{H}(a)$. When the collector groups t messages of matching id's, he can generate the corresponding private keys $d_{id} = Combine_{TIBE}(\mathsf{Pub}, VK, id, \theta_1, ..., \theta_t)$. With d_{id}, the collector can decrypt C using $Decrypt_{TIBE}$ and reveal the next hash bit for each message of the matching id set.

Assume for notational simplicity that $\{1, ..., t\}$ are the indices of t matching id's in contributions received by the collector. $\mathcal{C}$ uses the following algorithm to obtain the next hash bit by decrypting the next layer in each message. Each layer is the encryption of a triple except for the innermost, which is the original alert.

$BBDecrypt_{TIBE}(id, \{C_1, ..., C_t\}, \{\theta_1, ..., \theta_t\})$

1: $d_{id} = Combine_{TIBE}(\mathsf{Pub}, VK, id, \theta_1, ..., \theta_t)$

2: $(w, h) = id$

3: $ouput = \{\}$

4: **if** $length(h) = k$ **then**

5: **for** $i = 1$ to t **do**

6: $a_i = Decrypt_{TIBE}(\mathsf{Pub}, id, d_{id}, C_i)\}$

7: $output = output \cup \{a_i\}$

8: **end for**

9: **else**

10: **for** $i = 1$ to t **do**

11: $(id', C_i', \theta_i') = Decrypt_{TIBE}(\mathsf{Pub}, id, d_{id}, C_i)$

12: $ouput = output \cup \{(id', C_i', \theta_i')\}$

13: **end for**

14: **end if**

15: **return** *output*

As a result of the decryption process, alerts that were submitted by at least t contributors will be completely decrypted, because, for each id in the encryption, the threshold will be met. After the last layer, when all k bits have been decrypted by the use of $BBDecrypt_{TIBE}$, the original alert will be recovered. Figure 4.2 illustrates a single iteration of $Share_{TIBE}()$; Figure 4.3 illustrates the combination step done by $BBDecrypt_{TIBE}$. With overwhelming probability, messages that were not submitted by t contributors will not be completely decrypted. A message submitted by fewer than t contributors will be completely decrypted only if a collision in H occurs among the submitted alerts, and we asume that finding such a collision is infeasible for a computationally bounded adversary. However, *some* of the bits of such a hash value

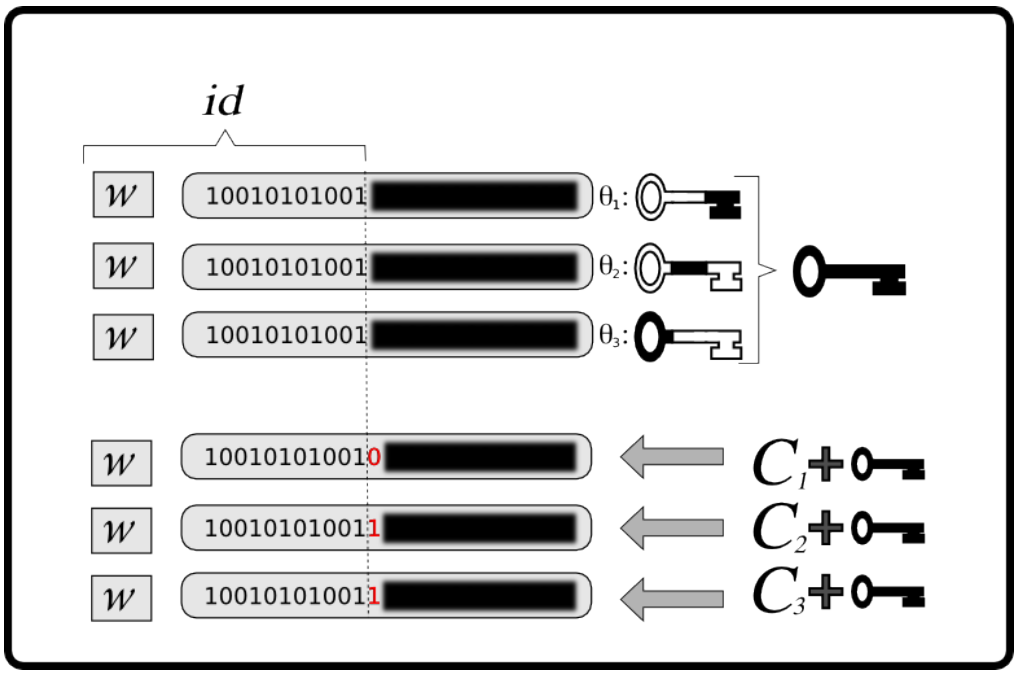


Figure 4.3: Illustration of $BBDecrypt_{TIBE}(id, \{C_1, ..., C_t\}, \{\theta_1, ..., \theta_t\})$ for $t = 3$. When the collector $\mathcal{C}$ groups t messages with matching id's, he can use the corresponding θ_i's to generate the complete private key for id. With that key, he can decrypt the next message to reveal the next hash bit.

h are likely to be revealed, because h is likely to have a common prefix with the hash values of $t-1$ other alerts. We call alerts whose hashes are partially but not completely revealed *non-decrypted alerts* and examine them more carefully in the next section.

4.4 Security

For a non-decrypted alert a, some bits of the hash $\mathsf{H}(a)$ can be revealed to the Collector. Specifically, the bits that are revealed are those in the longest prefix that is common to the hash values submitted by t contributors. We argue in this section that the expected number l of bits revealed is small, because the probability of a collision with t alerts decreases exponentially with the length of the prefix. In some

scenarios (e.g., one in which an adversary simply wants to learn one bit of information such as whether at least one contributor participated in a given operational period), this security guarantee is inadequate, but there are many scenarios (e.g., the IP-address reporting scenario discussed below) in which it is adequate.

By using the TIBE scheme in [3] to compute shares, we ensure that the only information revealed to the collector is a common prefix of the hash values. For a non-decrypted alert a, if l bits of the hash are revealed during the $BBDecrypt_{TIBE}$ process, then that alert is indistinguishable from all others with the same l-bit hash prefix. If the alert space is of size 2^q, then we expect that a non-decrypted alert will be indistinguishable from 2^{q-l} others.

If the adversary who attempts to guess the value for non-decrypted alerts has significant *a priori* knowledge, this level of indistinguishability can be inadequate. For example, if the adversary knows that the alert must take one of two possible values, it is likely he will be able to learn which one it is. At the other extreme, if the adversary has no *a priori* knowledge, and alerts are drawn uniformly from a large space, the adversary will learn very little from the results that our collector publishes.

The interesting (intermediate) case is the one in which the adversary has limited *a priori* knowledge about contributors' private inputs. To model this intermediate case, we will assume that contributors' private inputs have *high min-entropy*. Roughly speaking, this means that the adversary does not assign high probability to the event that a will be contributed, for any a in the alert space.

More precisely, for a random variable X that takes values from a space AS, the min-entropy $H_\infty(X)$ is defined as:

$$H_\infty(X) = \max_{x \in AS} \log(\mathbb{P}(X = x))$$

If the contributors' private inputs have high min-entropy, we can use the *Entropic Security* criterion defined by Dodis and Smith in [16].

Definition *(Entropic Security)* The probabilistic Map Y hides all functions of X with leakage ε, if, for every adversary $\mathcal{A}$, there exists an adversary $\mathcal{A}'$ such that, for all functions f:

$$|Pr[A(Y(X)) = f(X)] - Pr[A'() = f(X)]| \leq \varepsilon$$

The map $Y()$ is called (h, ε)-entropically secure if $Y()$ hides all functions of X with leakage ε whenever the min-entropy of X is at least h.

For each non-decrypted alert a, l hash bits may be revealed. Recall that $pre_l(x)$ is the l-bit prefix of x, and consider $\mathsf{H}_l(x) = pre_l(\mathsf{H}(a)))$. If we model the hash function H_l as a random oracle, we get the following security property:

Theorem 4.4.1. *If H_l is a mapping from M to $\{0,1\}^l$ chosen at random such that $\mathbb{P}(\mathsf{H}_l(x)) = y) = \frac{1}{2^l}$, then H_l is $(h, \frac{\epsilon}{8})$-entropically secure with probability at least $1 - \frac{1}{2^l}$ for $\epsilon = \sqrt{\frac{1}{2^{\frac{h}{2}-l+1}}}$ when $\frac{h}{2} \geq l$.*

Theorem 4.4.1 says that, for most choices of a random mapping H_l, an *arbitrary* adversary will not be able to learn *any* function of x from $\mathsf{H}_l(x)$, when x is drawn from a distribution with min-entropy at least h. In combination, this entropic-security property of an ideal hash function and the semantic security of the TIBE scheme ensure that a computationally bounded adversary with limited knowledge of individual sites' private inputs will only learn a bounded amount of information about non-decrypted inputs.

Proof. The structure of the proof is as follows. First, we will show that, for most choices of H_l, the statistical difference between the uniform distribution over $\{0,1\}^l$

and $\mathsf{H}_l(x)$ is small when x is drawn from a distribution with min-entropy at least h. This fact will follow from the *leftover hash lemma* [30] (also used in the work of Dodis and Smith [16]), which implies that any distribution over a finite set $\{0,1\}^l$ with collision probability[4] $(1+2\epsilon^2)/l$ has a statistical difference of at most ϵ from the uniform distribution. Combined with Theorem 2.1 in [16], this fact directly implies Theorem 4.4.1.

We start by looking at the probability of a collision between two elements, as done in [30]. That will give us a bound on the statistical difference between H_l and the uniform distribution. Let X be a random variable that takes values from the set Z according to a distribution $f(x)$; that is, $\mathbb{P}(X = x) = f(x)$, for all $x \in Z$. Assume further that the min entropy $H_\infty(X) = h$. Let W be the collision probability of X. Then, for any $y \in Z$:

$$W = \mathbb{P}(\mathsf{H}_l(x) = y) = \sum_{x \in Z} I_{\{\mathsf{H}_l(x)=y\}} f(x)$$

where $I_{\{\mathsf{H}_l(x)=y\}}$ is and indicator function such that:

$$I_{\{\mathsf{H}_l(x)=y\}} = \begin{cases} 1 & \text{if } \mathsf{H}_l(x) = y \\ 0 & otherwise \end{cases}$$

It is easy to see that $\mathbb{E}(W) = \frac{1}{2^l}$. Thus, in expectation, H_l behaves like a uniform distribution. We will show more generally that, in most cases, H_l, when applied to elements of a set Z with min-entropy h, behaves like a uniform distribution.

$$Var(W) = \sum_{x \in Z} f(x)^2 \frac{1}{2^l} \left(1 - \frac{1}{2^l}\right)$$

[4]The collision probability of a distribution D over set S is the probability that two independent draws from S according to D will result in the same value.

We know that

$$\sum_{x \in Z} f(x) = 1$$

and, from the fact that the min-entropy of X is h, that $f(x) \leq \frac{1}{2^h}$. Thus, we have that

$$Var(W) \leq \frac{1}{2^l}\left(1 - \frac{1}{2^l}\right)\frac{1}{2^h} \leq \frac{1}{2^{l+h}}$$

Using Chebyshev's inequality, we see that, for all $z > 1$ with probability $\geq 1 - \frac{1}{z^2}$,

$$W \leq \frac{1}{2^l} + \frac{z}{\sqrt{2^{l+h}}}$$

Choosing $z = 2^{\frac{l}{2}}$,

$$W \leq \frac{1}{2^l} + \frac{1}{2^{\frac{h}{2}}}$$

Using this bound as a worst case, we get a statistical difference of $\epsilon = \sqrt{\frac{1}{2^{\frac{h}{2}-l+1}}}$ between the distribution of $\mathsf{H}_l(X)$ and the uniform distribution.

From Theorem 2.1 in [16], we get that $\mathsf{H}_l(X)$ is $(h, \frac{\epsilon}{8})$-entropically secure when $\frac{h}{2} \geq l$. □

4.5 Efficiency

The total communication complexity of our protocol is $O(Q \cdot c \cdot n)$, where Q is the length of a share computed by the $Share_{TIBE}$ function specified in the previous section. Furthermore, the communication cost for each contributor is $O(Q \cdot c)$. These facts render our protocol more practical and scalable than that of Kissner and Song [33] and other protocols based on secure function evaluation, in which the total communication complexity grows quadratically with the number of participants, and the cost to each participant grows linearly with the number of participants.

Encryption $Share_{TIBE}()$ and decryption $BBDecrypt_{TIBE}$ are feasible yet expensive operations. $Share_{TIBE}()$ entails, for each alert, a constant k number of TIBE-encryption operations. Each TIBE encrypt/decrypt requires a small number of large-integer algebraic operations that are computationally intensive but feasible for today's commodity computers.

4.6 Experiments

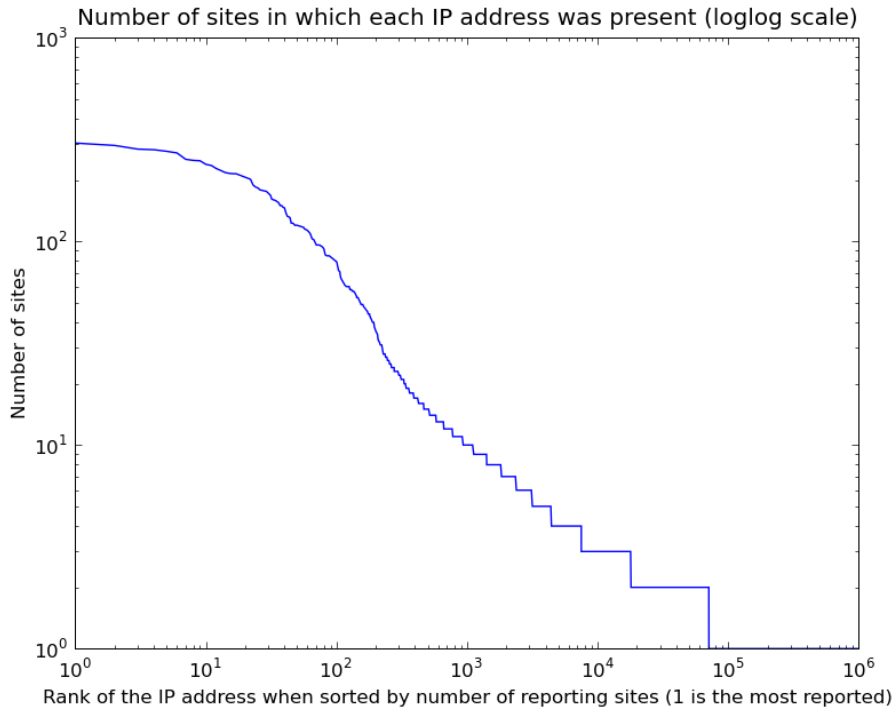
In this section, we provide experimental evidence that the conditions for our protocol to perform satisfactorily are met in the simple case of sharing IP addresses of offending hosts.

We perform a simulation using a D-Shield data set that consists of incoming *flows*. Each flow includes the source IP address, destination IP address, protocol, and time. Using these flow data, we simulated our protocol for one operational period lasting 24 hours. The data set for this operational period contains 13.869.101 flows representing incoming connections to 714 contributors from 997.138 external IP addresses. Figure 4.4 shows the number of sites that reported each IP address. Observe that most IP addresses are reported by only a few contributors; however, a few IP addresses are reported by a large fraction of the contributors. These large-scale offenders are the ones that our protocol would bring to light.

We simulated the protocol for different thresholds to obtain the number of decrypted IP addresses (Fugure 4.5 bottom). We also computed the average and standard deviation of the number of hash bits revealed per non-decrypted IP address (Figure 4.5 top).

Naturally, smaller thresholds are more interesting. However, as the threshold decreases, the probability of a prefix collision for non-decrypted elements increases,

Figure 4.4: Reported frequency: Number of sites that reported each IP address



and this raises the amount of information revealed. As an example, for a threshold of 30, about 200 of the million IP addresses get decrypted. At that level, almost all (99.99%) non-decrypted values reveal fewer than 17 bits. Because the IP-address space is almost fully used, we could expect connections from all the 2^{32} addresses. Therefore, each non-decrypted value is indistinguishable from approximately $2^{32-17} = 2^{15} = 32.768$ others. Toward the higher end of the range, for a threshold of 150, around 40 values get decrypted, and each non-decrypted value is indistinguishable from 262.144 others. IP addresses are a rather small alert space, providing at most 2^{32} alert values. On the other hand, IP addresses are not highly sensitive, and thus this level of protection is probably acceptable.

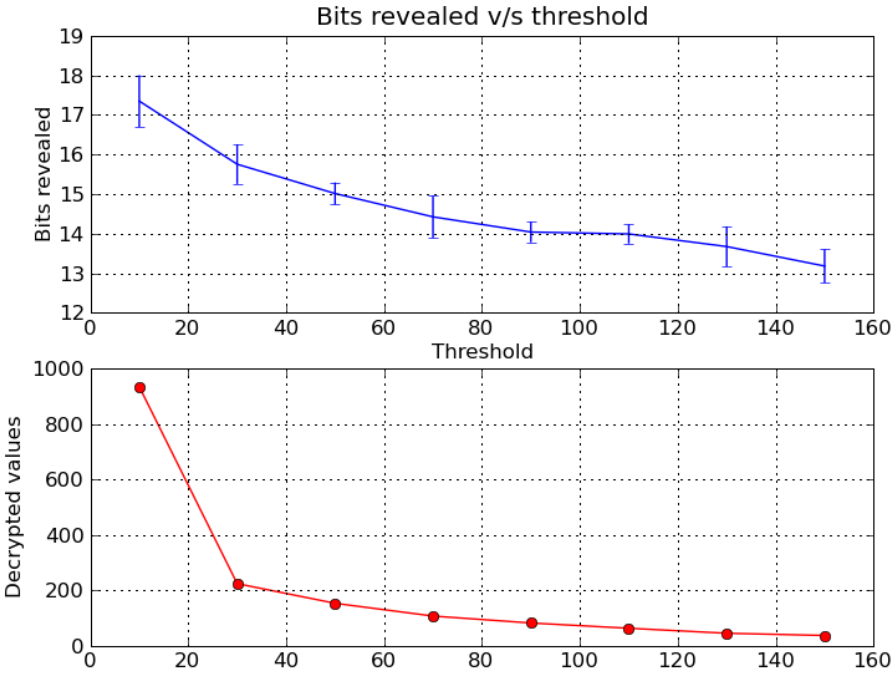


Figure 4.5: Top: Number of bits revealed (average with standard deviation bar). Bottom: Number of IP addresses successfully decrypted for each threshold

4.7 Malicious Contributors

So far, our security analysis has focused on protecting contributors from a collector who may want to discover sensitive alerts, *i.e.*, those that have not occurred at least t times in an operational period. However, contributors themselves may deviate from the protocol in ways that harm each other. Obviously, a registered contributor may choose simply not to participate in the protocol during a particular period. This should not pose a real problem if $t \ll n$ and network-side threats affect a large fraction of the participants.

A more serious concern is the potential for $t-1$ contributors to collude in order to harm another. Suppose that $\mathcal{S}_i$ learns about an alert a that is truly private to $\mathcal{S}_j$ – that is, $\mathcal{S}_j$'s security-monitoring system generates alert a for reasons that are unique to that site, but $\mathcal{S}_i$ (perhaps through organizational espionage or through hacking into $\mathcal{S}_j$'s network) obtains a copy of a. If $\mathcal{S}_i$ can convince $t-2$ other contributors to send in shares of a and sends in one itself, then the collector will publicize a, potentially revealing sensitive information about $\mathcal{S}_j$. Unfortunately, the ability of $t-1$ participants to collude successfully in an effort to expose or harm another is an inherent weakness of threshold-t security mechanisms. The antidote is basically to raise the threshold.

4.8 Conclusion and Open Problems

We have presented a novel, scalable protocol for threshold-union computation that achieves consensus privacy under plausible conditions. There are many natural directions for further work along these lines, and we give four of them here. First, it would be interesting to see how an alert-sharing system based on threshold union [55] performs in practice. Second, it would be worthwhile to develop other real appli-

cations of threshold union. Third, it is possible that we could assume something weaker than high min-entropy; under this assumption, we achieve entropic security against an arbitrary adversary for part of our construction, but our overall construction is only secure against a computationally bounded adversary, because we use a TIBE scheme. Finally, we would like to develop methods to test our assumption of high-entropy input data and believe that those testing methods would find other applications in network monitoring and security.

Chapter 5

Single-Database, Computationally Symmetric, Private Information Retrieval[5]

5.1 Introduction

5.1.1 Motivation

Picture the following scenario. Alice is looking for gold in California. What Alice does is look for a place with a little gold and follow the trace. Now, Alice wants to find gold in a place where no mining permit has been awarded, but many permits have been awarded in California during the gold rush. What Alice does is to walk around California with a GPS and a notebook computer. Whenever she finds a trace of gold, she follows it querying whether any permit has been awarded in that location. If she finds a trace of gold in a piece of land on which no permit has been issued, she can request the permit and start mining for gold. The problem is that she is worried

[5]This chapter is based on [52].

that Bob's Mining Inc., the service she queries about permits, might cheat on her. Because Bob knows she is looking for gold in California (Alice said so when signing up for Bob's service), he knows that, if she queries from some location, then there is gold there. So, if she queries a location and there is no permit awarded, Bob may run to the permit office and get the mining permit for that location. Depending on privacy and economic constraints, a few solutions come to mind. Alice might buy from Bob the whole database for California. Alice then can make all the queries to her own database, and Bob will never find out where Alice is looking for gold. But this might be very expensive, because Bob charges per query; what he charges for the whole database will probably be more than what Alice is willing to pay. Alice can also, with each query, perform a collection of fake queries so that Bob can't figure out which is the real query (this leaks information unless she queries the whole database!), but that still makes Alice pay for more queries that she would like.

This Alice-and-Bob scenario is a basic motivation for Private Information Retrieval (PIR): a family of two-party protocols in which one of the parties owns a database, and the other wants to query it with certain privacy restrictions and warranties. Since the PIR problem was posed, different approaches to its solution have been pursued. In this chapter, we first present the general ideas underlying the variations on and proposed solutions to the PIR problem. We then present a collection of basic building blocks that allow the implementation of a general-purpose PIR protocol. Finally, we present the specification and evaluation of a particular PIR protocol that we have implemented.

5.1.2 Overview of PIR

As mentioned in the motivation, the goals of PIR would be realized if Bob were to send the whole database to Alice. That would be satisfactory if Bob did not care

whether Alice learned anything except the answer to her query. In that situation, the challenge is to devise a protocol that reduces the amount of data Bob has to send in order for Alice to learn the answer to her query without Bob's learning what the query was. In general, that can only be done by having replicated, non-communicating databases. Going back to the Alice-and-Bob scenario, there is not one Bob but a collection of them with identical databases. In this way, the query can be hidden if Alice interacts with all the Bobs in such a way that each Bob is never sure whether the query he receives is the real one. The solutions proposed in this line of work achieve a lower number of queries (sublinear in the database's size) if more replicated Bobs are available. In this kind of solution, Alice's privacy is protected, and the objective is to reduce the number of queries needed. Most of the protocols in this line of work present solutions that are private from an information-theoretic point of view. For example, Chor et al. [7] show that, if the database is replicated two or more times, sublinear communication complexity can be obtained. Here, "sublinear communication" means that, in the execution of the protocol, the number of bits transferred is little-oh of the number of bits in the whole database. As mentioned above, the size of the whole database is a trivial upper bound on the communication complexity of the PIR problem. In the information-theoretic approach, the protocols are composed of many single-element queries, each of cost one, because exactly one element of the database is transferred in response to each query.

An important improvement in PIR was put forth by Kushilevitz and Ostrovsky [34]; they presented a PIR protocol that requires no replication. Their protocol, based on the hardness of the Quadratic Residuosity problem, is private from a computational complexity point of view; so, to distinguish it from the information-theoretical approach, it is known as cPIR, for "computational PIR". This idea was

first considered in [6].

One variation on the standard PIR scenario, in which only Alice's privacy is safeguarded, is the SPIR scenario (Symmetric PIR). In SPIR, we not only care about Bob's not learning anything about Alice's query, but we also want Alice not to learn anything about entries in Bob's database other than the one she queried. This is very similar to the one-out-of-N Oblivious Transfer problem, and, as we will see below, the two problems are closely related.

In this chapter, we focus on the implementation of a specific SPIR protocol proposed by Naor and Pinkas [41] that uses Oblivious Transfers as a building block. One of the properties of this protocol is that it requires only one initialization phase for a sequence of queries, thus amortizing the cost of the initialization phase. We will also show a variation of the protocol, proposed by Boneh, that eliminates the initialization phase by introducing a cPIR query as part of the protocol.

5.2 Building blocks

In this sections, we present a collection of protocols that are required for the SPIR implementation. In all of them, the Sender (Bob) owns a database, and the Receiver (Alice) wants to get the i^{th} value in this database.

In the protocols displayed below, "Via P" means that the value directly below is transmitted using a protocol for P.

5.2.1 One-out-of-two Oblivious Transfer: OT_1^2

In a one-out-of-two Oblivious Transfer, which we denote by OT_1^2, the Sender holds two values. As a result of the protocol, the Receiver learns one value of his choice and nothing about the other. The Sender learns nothing about the choice made by

the Receiver. We implemented a protocol by Naor and Pinkas [42], which proceeds as follows:

Initialization: The Sender and Receiver agree on a large prime q and a generator g for Z_q^*. In the actual implementation, the Sender generates them and sends them to the Receiver; the pair (q, g) can be used in several transfers. We have the Sender generate them, because we want the Receiver not to be able to compute the discrete log efficiently, and no extra information that enables him to do so is given as part of the protocol. $H(\cdot)$ is a random oracle (in practice, a hash function).

Receiver		Sender
		/Choose C uniformly at random from Z_q and send it to the Receiver/
		$C \leftarrow_r Z_q$
	$\leftarrow C$	
/Choose k uniformly at random from Z_q. Compute PK_σ and $PK_{1-\sigma}$. Send PK_0 to Sender/		
$k \leftarrow_r Z_q$		
$PK_\sigma = g^k$		
$PK_{1-\sigma} = \frac{C}{PK_\sigma}$		
	$PK_0 \rightarrow$	

/Compute PK_1. Choose r_0 and r_1, uniformly at random (and independently) from Z_q. Let E_0 and E_1 be the encryptions of M_0 and M_1, respectively. Send E_0 and E_1 to the Receiver./

$PK_1 = \frac{C}{PK_0}$

$r_0 \leftarrow_r Z_q$

$r_1 \leftarrow_r Z_q$

$E_0 =< g^{r_0}, H(PK_0^{r_0}) \oplus M_0 >$

$E_1 =< g^{r_1}, H(PK_1^{r_1}) \oplus M_1 >$

$\leftarrow< E_0, E_1 >$

/Compute M_σ/

$M_\sigma = H((g^{r_\sigma})^k) \oplus M_\sigma$

The size of M_i must be smaller than the size of the output of the hash function H. We used `sha-256` for H; thus, M_i can be up to 256 bits.

5.2.2 One-out-of-N Oblivious Transfer: OT_1^N

One-out-of-N Oblivious Transfer, which we denote by OT_1^N, is a generalization of OT_1^2. In OT_1^N, the Sender has a list of N elements instead of two. The desired properties are that the Receiver learn only the i^{th} value in the database and that the Sender learn nothing about i. Our implemented OT_1^N protocol is the following:

Initialization: The Sender holds values $X_1, X_2, \ldots, X_N$ with $X_i \in \{0,1\}^m$ and $N = 2^l$. The Receiver wants to learn X_i.

Receiver		Sender
		/Choose l pairs of keys $(K_1^0, K_1^1), (K_2^0, K_2^1), \ldots, (K_l^0, K_l^1)$ uniformly at random from $\{0,1\}^t$. Each K_j^b is a t-bit key to the pseudorandom function F_k./
		$(K_j^0, K_j^1) \leftarrow_r \{0,1\}^t \times \{0,1\}^t$ $j \in \{0, \ldots, l\}$
		/For all $1 \leq I \leq N$, let $(i_1, i_2, \ldots, i_l)$ be the bits of I. Compute Y_I/
		$Y_I = X_I \oplus \oplus_{j=1}^{l} F_{K_j^{i_j}}(I)$
		/Sender and Receiver engage in an OT_1^2 for the strings $< K_j^0, K_j^1 >$. /
	/Via OT/	
	$\leftarrow \{K_j^{i_j}\}$	
	$\leftarrow Y_1, .., Y_N$	
/Using the keys $\{K_j^{i_j}\}$ and $Y_1, .., Y_N$, compute the ouput value X_I./		
$X_I = Y_I \oplus \oplus_{j=1}^{l} F_{K_j^{i_j}}(I)$		

The paper by Naor and Pinkas that proposed this protocol [40] is ambiguous in defining the pseudorandom function F as a mapping F_K from $\{0,1\}^m$ to $\{0,1\}^m$ but using it to compute $F_k(I)$, where $1 \leq I \leq N$; in our implementation, we needed to use a pseudorandom function $F_K : \{0,1\}^t \rightarrow \{0,1\}^m$, where $t = \lceil log_2(N) \rceil$. Regarding the domain of this OT_1^N implementation, K_j^b will be the input to an OT_1^2; thus, the

X_i must be the same size as the output of the pseudorandom function.

5.2.3 PIR

In a PIR protocol, the Sender holds a database of size N. The Receiver wants the i^{th} value in this database. As a result of the protocol; the Receiver must learn the i^{th} entry in the database, but the Sender must learn nothing about i. In a general PIR protocol, by "learn nothing," we mean that a computationally unbounded Sender can learn nothing about i. That means privacy is preserved from an information-theoretic point of view. We mention this kind of protocol for clarity, but it is not used in our implementation. There is no restriction on what the Receiver can learn as a result of the protocol.

5.2.4 cPIR

A cPIR protocol is similar to a PIR protocol. The only difference is that privacy is safeguarded against a polynomially bounded Sender.

5.2.5 Additional tools

A few additional cryptographic techniques will be needed to implement the SPIR protocol.

5.2.5.1 Random Oracle

A random oracle is a protocol-design tool that gives all the parties in the protocol a common source of random bits. In practice, the shared randomness is provided by a cryptographically strong hash function like `sha-1`.

5.2.5.2 Sum-consistent synthesizer

A sum-consistent synthesizer is a function S for which the following holds:

- S is a pseudorandom synthesizer.
- For every X, Y and X', Y', if $X + Y = X'' + Y'$, then $S(X, Y) = S(X', Y')$.

where a pseudorandom synthesizer is basically a pseudorandom function on many variables that is pseudorandom on each one of them. Pseudorandom synthesizers where introduced by Naor and Reingold in [44] .

5.3 Our cSPIR implementation

The scenario in which an SPIR protocol is used is similar to that in which a cPIR protocol is used. However, at the end of the execution of an SPIR protocol, the Receiver should have learned nothing about values in the database other than the i^{th} one. Note that SPIR is the most constrained of all PIR variations and that an OT_1^N is an SPIR protocol.

We implemented a variation of version 3 of the cSPIR protocol presented in [41]. (A cSPIR protocol has both the symmetric-privacy property of an SPIR protocol and the polynomially-bounded-adversary property of a cPIR protocol.) Version 3 of the protocol is not secure, because after several queries (3 to be exact) it leaks information. The authors of [41] propose a high-cost fix. In our implementation, we lower the cost by modifying a step in the protocol. The original protocol as described in [41] is

Initialization: The Sender prepares $2\sqrt{N}$ random keys $(R_1, R_2, \ldots, R_{\sqrt{N}})$ and $(C_1, C_2, \ldots, C_{\sqrt{N}})$. For every pair $1 \leq i, j \leq \sqrt{N}$, he also prepares a commitment Y_{ij} of X_{ij} ($Y_{ij} = commit_{K_{ij}}(X_{ij})$) and sends them to the Receiver.

If the Receiver wants to learn X_{ij}, then the protocol is

Receiver		Sender
		/ Choose r_C uniformly at random and r_C such that $r_C + r_R = 0$. /
		$r_C \leftarrow_r \{0,1\}^t$
		$r_R \leftarrow -r_C$
		/Sender and Receiver engage in an $OT_1^{\sqrt{N}}$ protocol for the values $R_1 + r_R, R_2 + r_R, \ldots, R_{\sqrt{N}} + r_R$ /
	/ Via OT /	
	$\leftarrow R_i + r_R$	
		/Sender and Receiver engage in an $OT_1^{\sqrt{N}}$ protocol for the values $C_1 + r_C, C_2 + r_C, \ldots, C_{\sqrt{N}} + r_C$ /
	/ Via OT /	
	$\leftarrow R_j + r_C$	
$K_{ij} = S(R_i + r_R, C_j + r_C)$		
/Open the commitment Y_{ij}, and reveal X_{ij}/		
$X_{ij} = openCommitment_{K_{ij}}(Y_{ij})$		

In the modified version, if the Receiver wants to learn X_{ij}, then the protocol proceeds as follows:

Initialization: The Sender prepares $2\sqrt{N}$ random keys $(R_1, R_2, \ldots, R_{\sqrt{N}})$ and $(C_1, C_2, \ldots, C_{\sqrt{N}})$. For every pair $1 \leq i, j \leq \sqrt{N}$, he also prepares a commitment Y_{ij} of X_{ij} $(Y_{ij} = commit_{K_{ij}}(X_{ij}))$.

Receiver		Sender
	/ Via SPIR/	
	$\leftarrow Y_{ij}$	
		/ Choose r_C uniformly at random and r_C such that $r_C + r_R = 0$. /
		$r_C \leftarrow_r \{0,1\}^t$
		$r_R \leftarrow -r_C$
		/Sender and Receiver engage in an $OT_1^{\sqrt{N}}$ protocol for the values $R_1 + r_R, R_2 + r_R, \ldots, R_{\sqrt{N}} + r_R$ /
	/Via OT/	
	$\leftarrow R_i + r_R$	
		/Sender and Receiver engage in an $OT_1^{\sqrt{N}}$ protocol for the values $C_1 + r_C, C_2 + r_C, \ldots, C_{\sqrt{N}} + r_C$/
	/Via OT/	
	$\leftarrow R_j + r_C$	
/Open the commitment Y_{ij}, and reveal X_{ij}/		
$X_{ij} = openCommitment_{K_{ij}}(Y_{ij})$		

The main difference is that the original protocol has communication cost $\Omega(n)$ for initialization, i.e., to send the commitments. In the modified version, that is replaced by a PIR query. One can randomize the commitments in each step to solve the information-leakage problem. The price of this is $O(N)$ local computation, which is better for overall protocol efficiency than $\Omega(N)$ communication.

The *commit* function was implemented with the symmetric cryptosystem AES; $commit_{K_{ij}}(X_{ij}) = AES - ENC_{K_{ij}}(X_{ij})$. The Sum-consistent synthesizer S was implemented with the hash function `sha-256`, $S(A, B) =$ `sha-256`$(A + B)$.

5.4 Network Layer

All of these protocols involve two-party computations. To implement them, we needed a network layer. Because the implementation was done in Java, a natural choice would have been RMI. The problem with RMI is that it does not work well for the kind of message-driven description of protocols that appear in the cryptographic-research literature. That means that using RMI would force the code to be structured differently from the way protocols are specified. That may not seem important at first glance, but it makes a big difference when checking and going through code. It is much easier to go over a piece of code that looks like the protocol written in the original paper. For that reason, our network layer was implemented by means of messages. Another important feature of the network layer is that it requires support for nested calls. The SPIR implementation is built upon other two-party protocols; so, we needed to have persistent connection in all the nested protocol calls. For a simple solution that combined both requirements, we implemented the Network-Broker class, a symmetric class written over TCP sockets that allows the sending of serializable objects over a network in a way that is very natural for implementing protocols involving two parties. Here is a short code sample showing an object message passing through a pair of threads:

```
public void testSimpleCase()
                throws IOException, ClassNotFoundException{
    int port = 40000;
```

```
final NetObjectBrokerServer server =
                    new NetObjectBrokerServer(port);
final Integer sc = new Integer(900);
final Integer cs = new Integer(901);



Runnable r = new Runnable(){
    public void run() {
        try {
            server.accept();
            System.out.println("Got a connectoon");
            Integer cs1 = (Integer)server.waitObject();
            assertEquals(cs1.intValue(),cs.intValue());
            server.sendObject(sc);
            server.close();
        } catch (Exception e) {
            e.printStackTrace();
            fail();
        }

    }
};
// Server is waiting
new Thread(r).start();
NetObjectBrokerClient client =
                new NetObjectBrokerClient("localhost",port);
```

Dataset size	Query time (seconds)
4	20.2
12	65.1
20	90.4
28	116.2
36	160.5
44	198.9
52	234.7
60	265.0
68	1002.3

Table 5.1: Time that our implementation takes to answer a cSPIR query, as a function of the size of the dataset. The reason for the huge increase in time between a dataset of size 64 and one of size 68 is that the program was running up against the machine's memory limit (1500Mb); our attempt to test a dataset of size 72 crashed when it completely ran out of memory.

```
    System.out.println("Connected, sending obj");
    client.sendObject(cs);
    Integer sc1= (Integer)client.waitObject();
    assertEquals(sc1.intValue(),sc.intValue());
    client.close();
}
```

5.5 Conclusions

The main objective of the work in this chapter was to see how applicable PIR protocols are in practice. As shown in Table 5.1, our implementation is fast enough to be useful for medium-sized databases, e.g., for the Taulbee salary database considered in Chapter 6. However, it is not fast enough for large databases; nor is any other PIR implementation in the literature.

A typical database-oriented application would benefit from a few more features of the query engine. Interesting extensions, from that point of view, would include the

ability to query for existence of an entry and the ability to query for string-valued or non-sequential keys. That can be easily implemented if the Sender maintains two tables. More ambitiously, we would like to be able to compute joins in a private way without NM complexity, where N and M are the sizes of the joined tables. Finally, many optimizations can be done. One of them is to replace the basic implementation of OT_1^2 with a more efficient one.

Chapter 6

Implementation of a Privacy-Preserving Survey System

6.1 Introduction

In this chapter, we describe our system for conducting surveys while protecting the privacy of the survey participants. We use the Computing Research Association (CRA) Taulbee Survey [9] of faculty salaries in North American Computer Science departments as a concrete example in which there are real privacy concerns but in which participation is too large and uncoordinated for direct application of standard protocols for privacy-preserving distributed computation.

This work on privacy-preserving survey computation was done jointly with Raphael S. Ryger and reported in preliminary form in [20]. The parts of the project for which I was the primary contributor are presented in Sections 6.2 through 6.5 below. The parts for which Ryger was the primary contributor will be presented in detail in his PhD thesis [51]; we briefly summarize some of them here in order to provide background for Sections 6.2 through 6.5.

6.1.1 The Taulbee Salary Survey

Traditionally, each CRA-member department was asked to report the minimum, median, mean, and maximum salaries for each *rank* of academic faculty (full, associate, and assistant professors and non-tenure-track teaching faculty) and the number of faculty members at each rank. Member departments are divided into four *tiers*: the 12 top-ranked departments (Tier 1), departments ranked 13 through 24 (Tier 2), departments ranked 25 through 36 (Tier 3), and all other member departments (Tier 4). For each tier-rank pair, the survey published: (1) the minimum, maximum, and mean of the reported salary minima and maxima, (2) the mean of all salaries, and (3) the median of all salaries. Note that the exact median of the salaries could not be computed given the data traditionally reported by member departments; the median of means was used as an approximation.

More recently, the CRA has started asking member departments to provide complete, anonymized lists of salaries; here "anonymized" means that each salary figure is labeled by the rank but not the name of the corresponding faculty member. These data can be used by CRA to compute more accurate statistical data, such as the exact values of the median, other percentiles, and statistical variances. Here, the CRA plays the classical role of a *trusted third party*, i.e., the central node in the star-shaped communication pattern of a straightforward survey protocol — see Figure 6.1.

Not surprisingly, some departments voiced objections to the new form of the survey, citing legal issues preventing them from disclosing salary information or other privacy reservations. Member departments' objections are discussed more fully in [51]. It was precisely our anticipation of these objections that led to our undertaking this project.

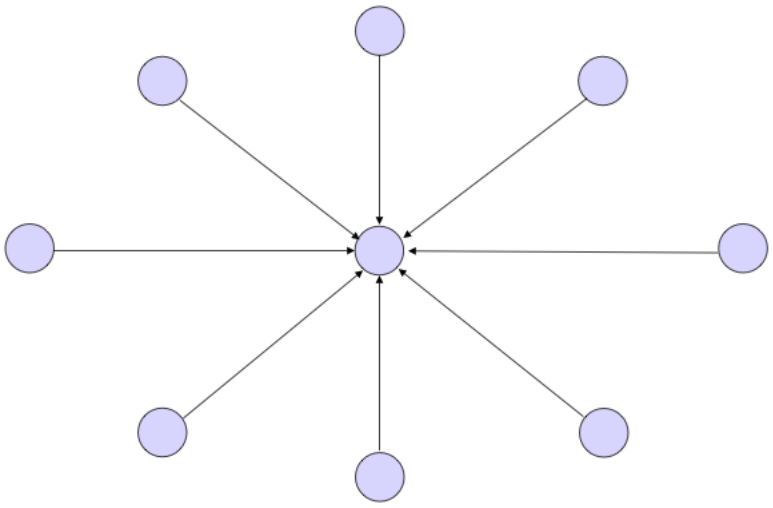


Figure 6.1: Communication pattern in classical trusted-party computations

6.1.2 Communcation Patterns and Protocol Structure

The natural approach to conducting this type of expanded Taulbee survey with privacy guarantees is to use general-purpose protocols for *secure multiparty computation* (SMPC). These are protocols that (under appropriate assumptions about computational complexity, the properties of the networked environment, and/or the fraction of participants who can be counted upon to follow instructions) enable a set $\{P_1, \ldots, P_N\}$ of N parties in possession of a set $\{x_1, \ldots, x_N\}$ of sensitive data items to compute a function $f(x_1, \ldots, x_N)$ in such a manner that all parties learn the result $y = f(x_1, \ldots, x_N)$, but no P_i learns anything about x_j, $i \neq j$, that is not inferrable from y and x_i; the protocols are "general-purpose" in the sense that they provide a procedure to transform an appropriate specification of *any* N-input function f into a specification of an SMPC protocol for N-party computation of f. An introduction to the theory of SMPC can be found in, *e.g.*, [27, 37].

Unfortunately, this straightforward approach has several major drawbacks. They

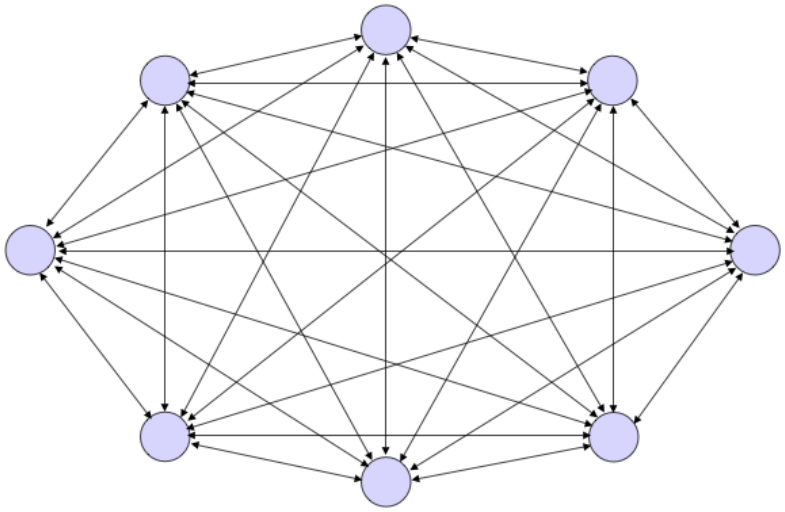


Figure 6.2: Communication pattern in general SMPC protocols

are discussed in detail in [51]; here, we confine ourselves to the observation that general-purpose SMPC protocols are highly interactive; see Figure 6.2. They require every party to be involved in multiple rounds of communication, each round depending on the messages that the other parties sent in the previous rounds. In the case of the Taulbee Survey, it is impractical to require the 12 busy department heads in any of the top 3 tiers to be online at the same time. For the fourth-tier computations, approximately 160 department heads would have to interact!

Our privacy-preserving survey-computation system extends earlier work on private auctions by Naor, Pinkas, and Sumner [43]. In particular, we adopt their approach of designating a small number M of parties, the *computation servers*, to do the main secure computation on behalf of the larger number N of end-user survey participants, the *input providers*. (See Figure 6.3.) Specifically, for reasons given in [51], we chose to use $M = 2$ computation servers; in practice, it would be natural for these roles to be played by organizations such as ACM, USENIX, or IEEE that exist to support the computing profession. This choice enabled us to build upon

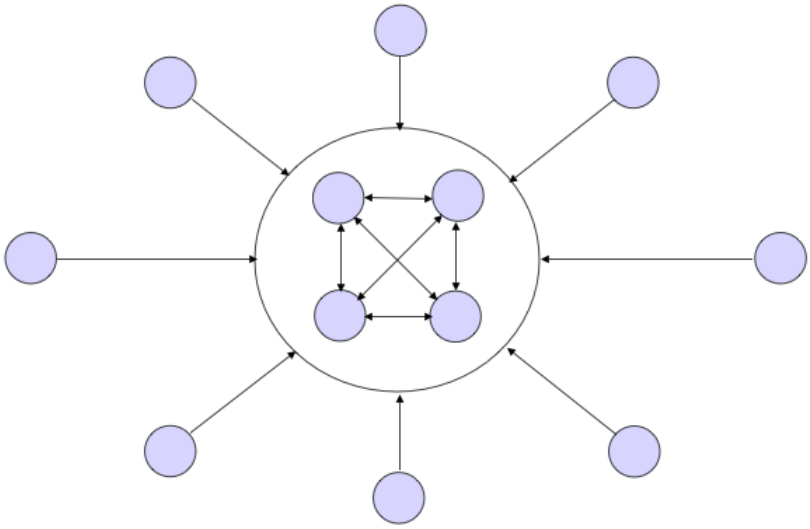


Figure 6.3: Communication pattern in M-for-N-Party SMPC

the Fairplay platform for general-purpose, secure, two-party computation [39]. The system supports an arbitrary number N of input providers. It is secure against any coalition of "honest but curious" adversaries that does not control both computation servers. The system is facilitated by a *control server* (to be run at CRA, presumably, for the Taulbee Survey) that provides administrative functions including registration of input providers, supervision of the submission of data, and launch of the core SMPC computation. Input providers (CS department heads or their delegates in the case of the Taulbee survey) and survey administrators (CRA in the Taulbee case) interact with the control and computation servers via Web interfaces.

6.1.3 XOR Shares

Our system and the Fairplay platform on which it is based use Yao's two-party, secure-computation protocol, which applies to functions represented as circuits with binary gates [72]. In preparation for the launch of the Yao two-party, secure com-

putation, our system must collect inputs, and it must generate the circuit that will accept the submitted inputs and compute the appropriate function.

Consider a circuit C with L input wires that computes the desired results of the survey, privacy concerns aside. The input of each input provider corresponds to a subset of the input wires of the circuit. Define a circuit C' with $2L$ input wires, where L of these input wires are *red*, and L of them are *blue*. The circuit C' is generated by taking every input wire of C and defining it to be the exclusive-or (XOR) of one red input wire and one blue input wire.

Collection of salary data proceeds as follows: Each input provider defines, for each of its input bits, two random bits (red and blue) whose XOR is equal to the input bit. It then sends all its red bits, together with the indices of the corresponding wires, to one of the computation servers and similarly sends all its blue bits to the other server. This is the only operation done by the input provider, and it can be done independently of other input providers. A database on the control server tracks submissions by input providers. At some point, at the discretion of the survey administrator, the control server instructs the two computation servers to engage in Yao's protocol to compute the output of C', which is equal to the output of C.

As explained by Ryger [51], our overall system architecture is novel in addressing the reality of haphazard input arrival (and possible non-arrival); indeed, "the function" to be computed securely is not known until the system decides at some point to cease collecting inputs (at which point neither the input providers nor their computers can be expected to be available for any interaction).

6.2 The User Interface

The user registers as a participant using a Web form. Once registered, she can log in, establishing a session to which the control server assigns an ID, and she can invoke another Web form into which she may enter her survey input, the faculty-salary list in the case of the Taulbee Survey. At this stage, the data are still local to her machine and are not submitted to the servers. When she clicks the "Submit" button, JavaScript code running in her browser generates "red" and "blue" shares for every input bit. The set of red bits and the user's session ID are sent to one computation server over an encrypted SSL channel, and the blue bits and the session ID are sent to the other server. *Note that the cleartext data never leave the client machine!* After receiving the data, the computation servers each notify the control server of the session ID and the number of data points received. The control server waits for the notifications to arrive from both computation servers, then records the data submission and sends an acknowledgment to the user. The user experiences all this as an an ordinary form-based interaction with "the Web site" for the survey.

6.3 The Function to be Evaluated Securely

The availability of the complete list of salaries enables the computation of more statistical data than was traditionally published in the Taulbee survey. In order to demonstrate the feasibility of the circuit-based solution, we use it to compute a sorted list of all the salaries submitted for each tier-rank pair. It is then a simple matter to change this circuit to output values such as the maximum and minimum, the median, quintiles, etc., as desired. Alternatively, the sorted list for the entire tier-rank may be deemed acceptable itself as output, considering that the very purpose of the survey in publishing various aggregate statistics is to convey just this distribution

in anonymized fashion. The latter approach admits arbitrary postprocessing of the tier-rank sorted lists, no longer requiring SMPC, to produce the statistics to be published. [6]

6.4 Circuit Generation

Fairplay receives a program written in a special high-level programming language, compiles it in a first stage into a circuit, and then, in a second stage, generates two Java programs that implement Yao's protocol for this circuit. Because our system computes a very specific family of circuits, we choose not to use the general-purpose Fairplay compiler, preferring to use our own dedicated circuit generation to keep the gate count down. The sorting we intend to do, like any algorithm that treats a large number of inputs symmetrically, is naturally expressed in high-level code that loops over array index values. Fairplay supports array indexing but does so in a general fashion that can handle indices not known until run time. Each instance of array indexing has a set of wires to hold the index value and circuitry to find the array element indexed by the value held in those wires. While this is a general approach, it is very wasteful in cases where the indices are known at compile time. We expect that Fairplay will evolve to allow efficient compile-time array indexing, not only by explicit constants but by what appear in the high-level language as assignment-proof loop-control variables. (All loops are unrolled in circuit generation, of course.) Furthermore, it is reasonable to expect that Fairplay will evolve to make certain specialized subcircuits of established utility, as for sorting, directly available to the high-level programmer in efficient implementations.

[6] The sorting can also be implemented using a mix network. However, the advantage of the circuit-based construction is that it can easily be adapted to output only the values in specific locations in the sorted list, *e.g.*, the items in the 10th, 50th, and 90th percentiles, without enumerating the other inputs.

Our circuit generator takes as input the number of inputs to the function to be evaluated and their length in bits. The output of our generator is a circuit that (a) reconstructs the individual bits of the original inputs (as entered by the participants) from the bit shares that the computation servers will provide as inputs to the circuit at run time, and (b) implements an odd-even sorting network for sorting the participants' reconstructed inputs. We feed the generated circuit to the second stage of the Fairplay system, which produces programs implementing secure computation for the circuit. We then run these Fairplay-generated programs to compute the sorted list of the inputs.

Note that, when handing the generated circuit to the Fairplay runtime system, we have a measure of freedom in designating which of the output wires from the sort we wish to have revealed as outputs of the secure evaluation of the circuit. We may request output of the whole sorted list or of any positionally defined sublist, e.g., quintiles.

The generated circuit begins with a layer of XOR gates each operating on a pair of corresponding bit shares from the two computation servers. The output of this first layer of gates comprises exactly the original values supplied by the survey participants. We now want to sort these values.

Not every sorting algorithm is suitable for implementation in a Boolean circuit. In general, branches of an algorithm are traversed or not depending on how the computation has gone up to some decision point, depending ultimately on the algorithm's input values. In contrast, a Boolean circuit is traversed in its entirety regardless of the input values. In quicksort, for example, the size of the sub-problems depends on the choice of the pivot; so the flow of the algorithm is input-dependent. On the other hand, bubble sort will do exactly the same comparisons regardless of the input; so bubble sort is a suitable algorithm, in the present sense, for implementation in

a Boolean circuit. The problem with bubble sort is that its $\Theta(n^2)$ cost is too high. Note that the Taulbee Survey involves hundreds of inputs, and we certainly would like our computational approach to be practical also for surveys that are larger by many orders of magnitude. Note also that, in our context, the secure evaluation of the algorithm itself introduces an additional large constant factor into the actual computational cost.

Sorting networks are a well-studied family of sorting algorithms that have just the property of input-independent flow that we need. They have this property because they are designed to run in evaluation circuits, similar to Boolean circuits, with a single genus of gate, a pair sorter that compares two inputs and outputs them in ascending order. We can implement an arbitrary-input pair sorter as a circuit in our model in which the wires are Boolean-valued. Sorting networks, then, actually target a more restrictive computational model than ours; so they are available for use in our SMPC context.

In applying the theory of sorting networks in our SMPC context, we must bear in mind a divergence in objectives. Sorting networks are envisioned as hardware resources, physical circuits in which all gates at a given depth may operate in parallel. Additional gates at a given depth constitute a one-time infrastructure cost, not a runtime cost. Accordingly, a primary performance objective in the network design is to minimize the overall circuit depth. In our setting, on the other hand, in which the circuit representation of the computation serves just as a framework for elaborate computations that will need to be carried out "at" each gate, an equally parallelized implementation would entail about as many full-fledged CPUs as the circuit has gates at a common depth, quite conceivably as many as half the number of (multi-bit) inputs to the circuit. We do not want to take for granted the availability of massively parallel computers; so we will be keenly interested in keeping the gate count

x_i	y_i	$knwn_{i-1}$	$rslt_{i-1}$	$knwn_i$	$rslt_i$
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	1	1

Table 6.1: Bit-i compare-gate (CMP)

$order$	x_i	y_i	x'_i	y'_i
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

Table 6.2: Bit-i swap-gate (SWP)

down for practical sequential computation. Small circuit depth will be a secondary consideration, relevant only to prospects of future parallelization.

To make sorting network theory available, we first implement the pair-sorter gate used in sorting networks as a compare-and-swap Boolean circuit with a pair of sets of Boolean-valued wires as input and as output, the multi-wire sets representing (up to a bound) the arbitrary input and output values of the pair-sorter gate. The implementation, naturally, first compares the two input sets, then feeds the comparison result with the two input sets to a set-swapping subcircuit.

The comparison of the input sets proceeds from the most (which we will number 1)

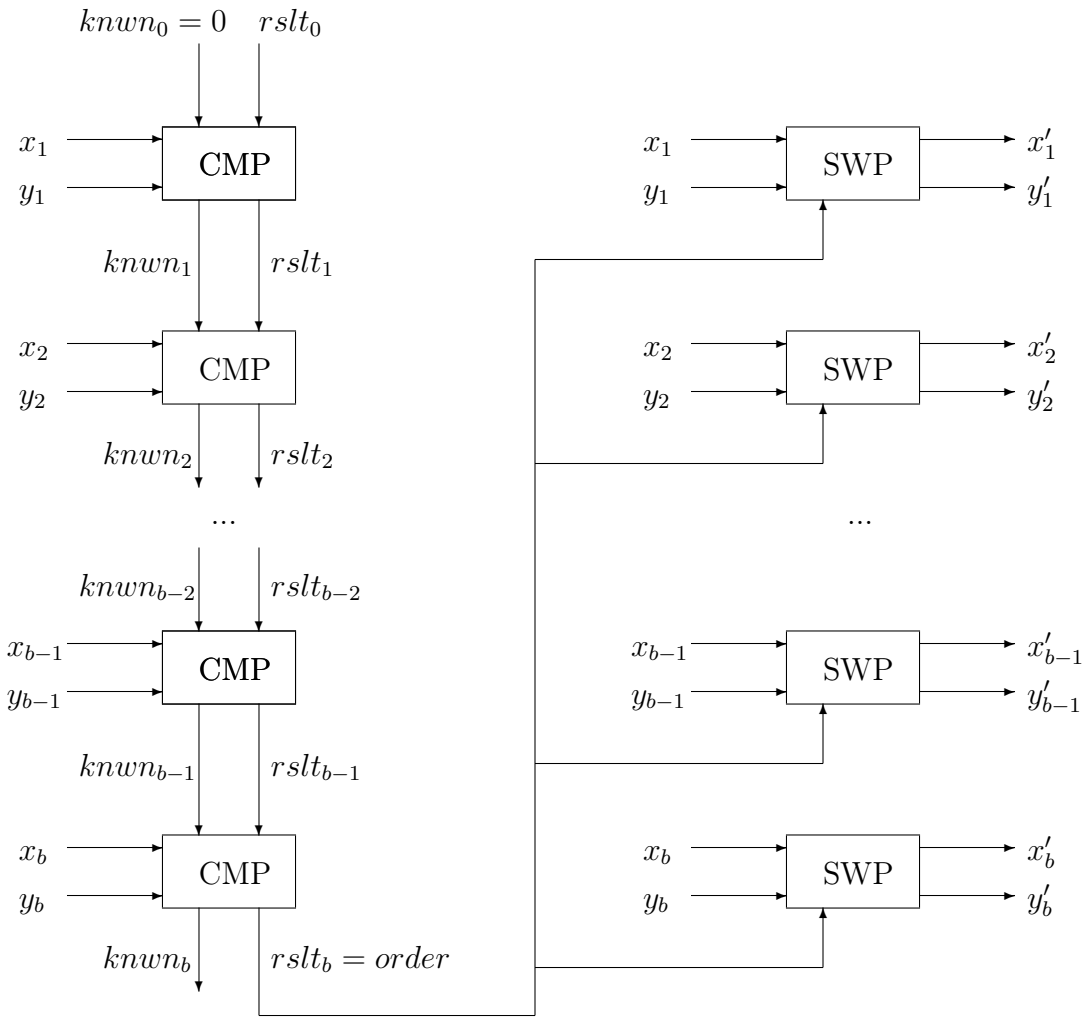


Figure 6.4: b-bit compare-and-swap Boolean circuit

to the least significant wire, cascading two additional binary values, $knwn_i$, indicating whether the result is already known from higher-order bit positions, and $rslt_i$, the comparison result (thus far). $knwn_0$ must be initialized as 0. (The initialization of $rslt_0$ does not matter.) The comparison at bit i takes as input the respective bit-i values, x_i and y_i, as well as $knwn_{i-1}$ and $rslt_{i-1}$ from the comparison at bit $i-1$. The gate table for the bit-i comparator is given in Table 6.1. Where b is the number of bits in the input and output sets, the $rslt_b$ output of the bit-b (last) comparator is the overall comparison result for the two input sets, taking value 0 if $x < y$, 1 if $y < x$, and, irrelevantly, $rslt_0$ if $x = y$.

The swapping (or not) of the input sets can proceed for each bit position independently of the others, controlled by the *order* bit value that was the $rslt_b$ output of the comparison subcircuit. Actual swapping occurs if $order = 1$. Table 6.2 gives the gate table for the bit-i swapper.

The assembly of the bit-comparison and bit-swapping gates into the desired multi-bit compare-and-swap Boolean circuit is illustrated in Figure 6.4.

Once the basic compare-and-swap circuitry is available, any sorting network can be implemented as a Boolean circuit. Bubble sort is very easy to implement in this structure, but, as we have said, it does not provide adequate performance, and so a better sorting network is needed. $O(n \log(n))$ (comparator count) sorting networks are known but are comparatively inefficient in applications of the scale we anticipate, due to large constant factors, despite their attractive asymptotic complexity. We choose instead to use an odd-even sorting network with a comparator-count complexity of $O(n \log^2 n)$ (see Batcher [1]). This turns out to be reasonably simple to implement, with performance significantly better than that of bubble sort, enabling our system to process large enough imput sets to make it useful. The odd-even sorting network is similar to merge sort but for sorting networks. Two steps are involved

in an odd-even sorting network: sorting and merging. We will assume that the input size is $n = 2^k$ for some k. The OddEvenMerge and OddEvenSort algorithms follow.

OddEvenMerge

Input: $\langle a_1, \ldots, a_n \rangle$ where $\langle a_1, \ldots, a_{n/2} \rangle$ and $\langle a_{n/2+1}, \ldots, a_n \rangle$ are sorted.
Output: $\langle a'_1, \ldots, a'_n \rangle$, the n input values sorted.

1: **if** $n = 1$ **then**
2: $\langle a_1 \rangle$
3: **else**
4: OddEvenMerge($\langle a_1, a_3, \ldots, a_{n/2+1}, a_{n/2+3}, \ldots \rangle$)
5: OddEvenMerge($\langle a_2, a_4, \ldots, a_{n/2+2}, a_{n/2+4}, \ldots \rangle$)
6: CompareAndSwap(a_i, a_{i+1}) for $i = 1, 3, \ldots, n-3$
7: **end if**

OddEvenSort

Input: $\langle a_1, \ldots, a_n \rangle$
Output: $\langle a''_1, \ldots, a''_n \rangle$, the n input values sorted

1: **if** $n = 1$ **then**
2: $\langle a_1 \rangle$
3: **else**
4: $\langle a'_1, \ldots, a'_{n/2} \rangle$ = OddEvenSort($\langle a_1, \ldots, a_{n/2} \rangle$)
5: $\langle a'_{n/2+1}, \ldots, a'_n \rangle$ = OddEvenSort($\langle a_{n/2+1}, \ldots, a_n \rangle$)
6: OddEvenMerge($\langle a'_1, \ldots, a'_n \rangle$)
7: **end if**

It remains only to generate the two files that the Fairplay runtime system needs to proceed, one specifying the circuit and the other describing the input and output wires (how the wires are grouped into inputs and outputs, who is to provide the inputs, who is to get the outputs). These are simple text files with straightforward syntax. The rest of the work is then done by Fairplay and the protocol code it generates.

6.5 Cost and Performance

When measuring the cost of the evaluation of the described function, we are interested in both time and space, which are closely related in a Boolean circuit. In our experience so far, it is space that limits the size of the problems that can be solved. For example, if the computation is executed in a typical current desktop computer with a 2 GHz CPU and 512 MB of RAM, the circuit can be evaluated in a few minutes for 250 inputs of 10-bit numbers, but it cannot go much further than that without running out of memory. The Fairplay internal object representation of the circuit to be securely evaluated is too big. The problem is multiplied many-fold if the cut-and-choose strategy to detect one mode of cheating is actually used. This strategy requires the generation of multiple instances of the circuit.

We estimate that a pair of state-of-the-art servers can handle about 800 inputs of the same length. The fact that the computation takes several minutes is not a problem, because it is done off-line, with no user waiting for the output.

The size of the circuit is dominated by the cost of the sorting stage. It is about $3N \ln^2 N \cdot \ell$ gates, where N is the number of inputs, and ℓ is the number of bits of a single input. All other stages are linear in $\ell \cdot N$.

6.6 Conclusion and Open Problems

From a technical point of view, the results of this project are fairly encouraging. Contrary to the prevailing belief in many quarters, it appears that secure, multiparty computation protocols that have been developed in the cryptography-research community could be deployed and used with a reasonable amount of effort. The Fairplay platform and FairplayMP [2], a more recent multiparty version thereof, are useful starting points for practical deployment. There are myriad promising directions for

further research on practical, secure, multiparty computation, some of which will be presented in [51]. Here, we mention just two that are directly relevant to our implementation and to the Taulbee-survey application.

First, the implementation could be improved. For example, one could improve the circuit implementation of the sorting network and, in particular, use different kinds of sorting networks and optimize the network size for inputs whose length is not a power of two. One could also improve the memory consumption of the implementation, because this is its major bottleneck. In particular, because we know that gates in the circuit are used in a certain order, we might be able to force the memory allocation dynamically to correspond to the set of gates that are about to be evaluated, rather than allocating memory to all gates.

Second, and more generally, if privacy-preserving protocols as the cryptography-research community conceives of them are to be widely used in practice, some non-technical barriers to adoption will have to be overcome. For example, as we learned from our unsuccessful attempt to have our implementation used for the expanded Taulbee survey, both survey conductors and survey participants are reluctant for non-technical reasons to adopt this type of system. The CRA, which has collected cleartext data about its member departments for many years in order to compile the Taulbee surveys, views its datasets as valuable assets and cannot be expected to do without them unless it faces a crippling non-participation rate. Member departments, including our own department at Yale, face institutional policies against disclosure of complete salary sets that do *not* distinguish between disclosing a dataset to a remote third party and disclosing it to a privacy-preserving, client-side program that splits the data items into XOR shares before transmitting anything to a remote server. They cannot be expected to adopt systems such as ours until there is an official legal basis for the position that these systems comply with the institution's policies on

data confidentiality. We elaborate further on barriers to adoption in [19].

Chapter 7

Conclusions and Future Directions

7.1 Conclusions

Contrary to the widespread perception that privacy-preserving protocols developed by the cryptography-research community are impractical, we have been able to develop prototype versions of several of them with reasonable amounts of effort. As explained in the Conclusions sections of the foregoing chapters, our prototype implementations would need to be more efficient to be widely usable, but it is quite plausible that the required efficiency improvements could also be accomplished with reasonable amounts of effort.

We are less sure about the plausibility of overcoming other types of obstacles. In particular, overcoming the institutional and legal barriers that we encountered in the Taulbee survey project will require an entirely new approach to potential users of SMPC protocols. In the web-search domain, PWS hides basically all of the information about the client machine that can be hidden without degrading search accuracy. However, neither PWS nor any other available browser plug-in hides the semantics of the user's queries which, unfortunately, may implicitly reveal his or her

identity. Moreover, as observed in Chapter 3, client-side privacy tools that rely on Tor impose performance costs that users are not willing to pay. For these and other reasons, we believe that radically different approaches and attitudes to user privacy should be considered; we describe one of them in the next Section.

7.2 Future Directions

It has been 34 years since the seminal paper of Diffie and Hellman [12] launched the era of "modern cryptography." During this era, many of the world's most decorated computer scientists have worked in the areas of cryptography and security and, needless to say, have made immense technical progress. Moreover, significant technical progress continues to this day, witnessed, for example, in the theoretical realm by Gentry's recent solution to the longstanding problem of fully homomorphic encryption [26] and, in the experimental realm, by continued development of tools such as the Fairplay platform [39, 2]. We can expect continued technical progress for many years to come; as noted above, the Conclusions sections of the five foregoing Chapters contain some specific technical directions that extend the work we have presented in this thesis and stand a reasonable chance of being tractable.

Yet, 34 years and counting of technical progress by the cryptography and security communities have not produced a networked world in which sensitive data are handled in a manner that is satisfactory to data subjects, data owners, data users, or any other legitimate constituency. Personal data are misused, confidential corporate data are disclosed, copyrights are infringed, and databases owned by one organization are illegally accessed by members of another. It is difficult to believe that further technical progress along already established lines will improve the situation fundamentally. In particular, client-side defenses (of which PWS is an example) may be an inherently

inadequate response to the problem of sensitive data in a networked world. Ideally, this class of tools would give users perfect control over whether and when their sensitive data are transmitted over a network. By design, such tools cannot prevent users from sharing data with others when these users genuinely want to share it. Indeed, the role of computers and networks in daily life has grown so markedly precisely because people and organizations *want* to share information and often benefit from doing so. In an environment in which massive amounts of sensitive information is deliberately shared, it may be inevitable that some of it is subsequently used for purposes that were not foreseen (and would not have been approved) by the people who shared it.

For this reason, some security and privacy researchers (see, e.g., [35, 70]) have recently advocated a paradigm shift: Rather than aiming primarily to *prevent* the transmission of sensitive data to parties who may misuse it, our community should accept the fact that these data will be widely transmitted and strive to hold all parties that receive them *accountable* for how they are used. Weitzner et al. [70] make the case as follows:

> For too long, our approach to information-protection policy has been to seek ways to prevent information from "escaping" beyond appropriate boundaries, then wring our hands when it inevitably does. This hide-it-or-lose-it perspective dominates technical and public-policy approaches to fundamental social questions of online privacy, copyright, and surveillance. Yet it is increasingly inadequate for a connected world where information is easily copied and aggregated, and automated correlations and inferences across multiple databases uncover information even when it is not revealed explicitly. As an alternative, accountability must become a primary means through which society addresses appropriate use.

We endorse the accountability agenda as a necessary complement to established research directions in security and privacy.

Bibliography

[1] Kenneth E. Batcher. Sorting networks and their application. In *Proceedings of the 32nd Spring Joint Computer Conference*, pages 307–314, Washington DC, 1968. Thomson.

[2] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: A system for secure multi-party computation. In *Proceedings of the 15th Conference on Computer and Communications Security*, pages 257–266, New York, 2008. ACM Press.

[3] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In *CT-RSA*, volume 3820 of *Lecture Notes in Computer Science*, pages 226–243, Berlin, 2006. Springer.

[4] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. *Journal of the ACM*, 48(4):702–722, 2001.

[5] Shuchi Chawla, Cynthia Dwork, Frank McSherry, Adam Smith, and Hoeteck Wee. Toward privacy in public databases. In *Proceedings of the 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 363–385, Berlin, 2005. Springer.

[6] Benny Chor and Niv Gilboa. Computationally private information retrieval. In

Proceedings of the 29th Symposium on Theory of Computing, pages 304–313, New York, 1997. ACM Press.

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

[8] Jeremy Clark, Paul van Oorschot, and Carlisle Adams. Usability of anonymous web browsing: an examination of tor interfaces and deployability. In *SOUPS '07: Proceedings of the 3rd Symposium on Usable Privacy and Security*, pages 41–51, New York. ACM Press.

[9] CRA Taulbee Survey. `http://www.cra.org/statistics/`.

[10] Yvo G. Desmedt. Some recent research aspects of threshold cryptography. In *Proceedings of the 1st International Workshop on Information Security*, volume 1396 of *Lecture Notes in Computer Science*, pages 158–173, Berlin, 1998. Springer.

[11] Yvo G. Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology – Crypto'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Berlin, 1990. Springer.

[12] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(5):644–654, 1976.

[13] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In *Fifth Workshop on the Economics of Information Security*, 2006. `http://weis2006.econinfosec.org`.

[14] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-

generation onion router. In *Proceedings of the 13th Security Symposium*, pages 303–320, Berkeley, 2004. USENIX.

[15] Roger Dingledine and Steven J. Murdoch. Performance improvements in tor or, why tor is slow and what we're going to do about it. `https://svn.torproject.org/svn/tor/trunk/doc/roadmaps/2009-03-11-performance.pdf`.

[16] Yevgeniy Dodis and Adam Smith. Entropic security and the encryption of high entropy messages. In *Proceedings of the 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 556–577, Berlin, 2005. Springer.

[17] EFF, AOL's massive data leak. `http://www.eff.org/Privacy/AOL/`.

[18] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the 22nd Symposium on Principles of Database Systems*, pages 211–222, New York, 2003. ACM Press.

[19] Joan Feigenbaum, Benny Pinkas, Raphael S. Ryger, and Felipe Saint-Jean. Some requirements for adoption of privacy-preserving data mining. PORTIA White Paper, 2005. `http://crypto.stanford.edu/portia/pubs/articles/FPRS446165839.html`.

[20] Joan Feigenbaum, Benny Pinkas, Raphael S. Ryger, and Felipe Saint-Jean. Secure computation of surveys. In *Proceedings of the EU Workshop on Secure Multiparty Protocols*, 2004. `http://www.zurich.ibm.com/~cca/smp2004/`.

[21] Edward Felten and Michael Schneider. Timing attacks on web privacy. In

Proceedings of the 7th Conference on Computer and Communications Security, pages 25–32, New York, 2000. ACM Press.

[22] Pierre-Alain Fouque and Jacques Stern. One round threshold discrete-log key generation without private channels. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 300–316, Berlin, 2001. Springer.

[23] Foxtor. `http://cups.cs.cmu.edu/foxtor/`.

[24] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology – Crypto'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Berlin, 1985. Springer.

[25] William Gasarch. A survey on private information retrieval. *Bulletin of the EATCS*, 82:72–107, 2004.

[26] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Symposium on Theory of Computing*, pages 169–178, New York, 2009. ACM Press.

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

[28] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[29] Google. `http://www.google.com/`.

[30] Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *Proceedings of the 30th Symposium on Foundations of Computer Science*, pages 248–253, Los Alamitos, 1989. IEEE Computer Society Press.

[31] Collin Jackson, Andrew Bortz, Dan Boneh, and John C. Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th International World Wide Web Conference*, pages 737–744, New York, 2006. ACM Press.

[32] Jap. `http://anon.inf.tu-dresden.de/index_en.html`.

[33] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *Advances in Cryptology – Crypto'05*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257, Berlin, 2005. Springer.

[34] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *Proceedings of the 38th Symposium on Foundations of Computer Science*, pages 364–373, Los Alamitos, 1997. IEEE Computer Society Press.

[35] Butler Lampson. Usable security: How to get it. *Communications of the ACM*, 52(11):25–27, 2009.

[36] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.

[37] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):59–98, 2009.

[38] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering*, page 24, Los Alamitos, 2006. IEEE Computer Society Press.

[39] Dalia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay—a secure two-party computation system. In *Proceedings of the 13th Symposium on Security*, pages 287–302, Berkeley, 2004. USENIX.

[40] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the 31st Symposium on Theory of Computing*, pages 245–254, New York, 1999. ACM Press.

[41] Moni Naor and Benny Pinkas. Oblivious transfer with adaptive queries. In *Advances in Cryptology – Crypto'99*, volume 1666 of *Lecture Notes In Computer Science*, pages 573–590, Berlin, 1999. Springer.

[42] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the 12th Symposium on Discrete Algorithms*, pages 448–457, New York/Philadelphia, 2001. ACM/SIAM.

[43] Moni Naor, Benny Pinkas, and Reuben Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st Conference on Electronic Commerce*, pages 129–139, New York, 1999. ACM Press.

[44] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random function. In *Proceedings of the 36th Symposium on Foundations of Computer Science*, pages 170–181, Los Alamitos, 1995. IEEE Computer Society Press.

[45] Jakob Nielsen. *Usability Engineering*, chapter 5. Morgan Kaufmann, San Francisco, 1993.

[46] p0f. `http://lcamtuf.coredump.cx/p0f/README`.

[47] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[48] Privoxy. `http://www.privoxy.org`.

[49] Vibhor Rastogi, Dan Suciu, and Sungho Hong. The boundary between privacy and utility in data publishing. In *Proceedings of the 33rd Very Large Data Bases Conference*, pages 531–542, New York, 2007. ACM Press.

[50] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, 1998.

[51] Raphael S. Ryger. *Toward Practical Large-Scale Secure Multiparty Computation*. PhD Thesis, Yale Computer Science Department, New Haven, to appear.

[52] Felipe Saint-Jean. Java implementation of a single-database computationally symmetric private information retrieval (cspir) protocol. Technical Report YALEU/DCS/TR-1333, Yale Computer Science Department, New Haven, 2005.

[53] Felipe Saint-Jean and Joan Feigenbaum. Usability of browser-based tools for web-search privacy. Technical Report YALEU/DCS/TR-1424, Yale Computer Science Department, New Haven, 2010.

[54] Felipe Saint-Jean, Aaron Johnson, Dan Boneh, and Joan Feigenbaum. Private web search. In *WPES '07: Proceedings of the 6th Workshop on Privacy in Electronic Society*, pages 84–90, New York, 2007. ACM Press.

[55] Felipe Saint-Jean, Jian Zhang, Joan Feigenbaum, and Phillip Porras. Privacy-preserving discovery of consensus signatures. Technical Report YALEU/DCS/TR-1429, Yale Computer Science Department, New Haven, 2010.

[56] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical Report SRI-CSL-98-04, SRI Computer Science Laboratory, Palo Alto, 1998.

[57] Simile. http://simile.mit.edu/java-firefox-extension/.

[58] Snort. http://www.snort.org/.

[59] SRI International. Cyber-threat analytics project. http://www.cyber-ta.org.

[60] Latanya Sweeney. K-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems*, 10(5):557–570, 2002.

[61] Tor. http://tor.eff.org.

[62] Torbutton. http://freehaven.net/~squires/torbutton/.

[63] Torbutton-download. http://freehaven.net/~squires/torbutton/.

[64] Torbutton-options. http://www.torproject.org/torbutton/options.html.en.

[65] Trackmenot. http://mrl.nyu.edu/~dhowe/TrackMeNot/.

[66] Triviadatabase. http://www.triviadatabase.com/.

[67] Johannes Ullrich. DShield home page. http://www.dshield.org.

[68] Vassilios S. Verykios, Elisa Bertino, Igor N. Fovino, Loredana P. Provenza, Yücel Saygin, and Yannis Theodoridis. State-of-the-art in privacy preserving data mining. *ACM SIGMOD Record*, 33(1):50–57, 2004.

[69] Vidalia. `http://www.torproject.org/vidalia/`.

[70] Daniel J. Weitzner, Hal Abelson, Tim Berners-Lee, Joan Feigenbaum, James Hendler, and Gerald Jay Sussman. Information accountability. *Communications of the ACM*, 51(6):82–88, 2008.

[71] Andrew C. Yao. Protocols for secure computation. In *Proceedings of 23rd Symposium on Foundations of Computer Science*, pages 160–164, Los Alamitos, 1982. IEEE Computer Society Press.

[72] Andrew C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Symposium on Foundations of Computer Science*, pages 162–167, Los Alamitos, 1986. IEEE Computer Society Press.

Appendices

Appendix A

User study trivia questions

What follows is the list of trivia questions used in the work described in Chapter 3, together with the answers that users were supposed to find on the web. Items are presented in the form Q | A, where Q is the question, and A is the answer.

In "A Clockwork Orange", who was Alex's favorite composer? | Ludwig Von Beethoven

Who wrote "The Crucible"? | Arthur Miller

Who wrote the book, "Dr. Jekyll and Mr. Hyde"? | Robert Louis Stevenson

What artist painted "Les Demoiselles d'Avignon"? | Pablo Picasso

In what city does Louis get interviewed in, in "Interview With The Vampire"? | San Francisco

In "Flowers for Algernon", who is Algernon? | The Mouse

Who wrote the book, "Pride and Prejudice"? | Jane Austen

Who wrote the book "Fowl Tips"? | Wade Boggs

What artist was on the cover of "Time" and "Newsweek" in 1975? | Bruce Springsteen

"The Lord of The Rings" was written by whom? | J.R.R. Tolkien

A person who poses for a painter is usually called a what? | Model

Famous painter Picasso went by the first name of? | Pablo

Finish the title of this book: "Of Mice And ..."? | Men

What is the name of the snowy owl that Hagrid bought for Harry Potter? | Hedwig

What was in the Trojan Horse? | Soldiers

Who wrote the "Cat in the Hat"? | Dr. Seuss

Who writes the "A Series of Unfortunate Events" book series? | Lemony Snicket

Who wrote the book "Mommie Dearest"? | Christina Crawford

Who wrote the book "The Rainmaker"? | John Grisham

Who wrote the book "The Partner"? | John Grisham

Who wrote the book "Tuesdays with Morrie"? |

Mitch Albom

Who did Rosie O'Donnell play in the Broadway show "Grease"? | Rizzo

Which Broadway musical featured the songs of Billy Joel? | Movin' Out

Which Steve Martin film was turned into a musical in 2005? | Dirty Rotten Scoundrels

Revived in 2002, "Man of La Mancha" is based on which novel? | Don Quixote

Which play won both the Pulitzer and Tony for Best Play in 2001? | Proof

Which musical legend is Liza Minnelli's mother? | Judy Garland

What city was Edgar A. Poe in when he died? | Baltimore

What does NHL stand for? | National Hockey League

What does M.V.P. stand for? | Most Valuable Player

Who wrote the 1959 book "Hawaii"? | James A. Michener

In 1956, J. R. R. Tolkien wrote which classic? | Lord Of The Rings

In 1937, John Steinbeck wrote which classic? | Of Mice And Men

Famous author Steinbeck has a full name of? | John Steinbeck

Famous author Updike has a full name of? | John Updike

What town is named after the author of "Last of the Mohicans"? | Cooperstown

Which of the following movies was not adapted from a James Clavell novel? | Gunga Din

What does the T.S. stand for in T.S. Eliot's name? | Thomas Stearns

Who killed Macbeth in the play "Macbeth"? | Macduff

What novel introduced the noun "Droogies"? | A Clockwork Orange

Who wrote "The Time Machine"? | H.G. Wells

Which poet wrote the poem "The Road not Taken"? | Robert Frost

Who wrote the book, "The Greene Murder Case"? | S. S. Van Dine

Who wrote the book "Gone with the Wind"? | Margaret Mitchell

Who is the author of "Angels & Demons"? | Dan Brown

Where does James have his adventure in "James and the Giant Peach"? | A Giant Peach

In Greek Mythology, who preceded the rising of the sun each morning? | Eos

Which Goddess was the Patron Saint of Athens? | Athena

A milk punch is made up of milk, sugar and which of the following? | Rum

What gives a "Brain" its blood vessels? | Grenadine

What is in the shot "Liquid Cocaine"? | Goldschlager

Which beer is brewed in the "land of sky blue waters"? | Hamm's Beer

Adolphus Busch started Anheuser-Busch in which U.S. city? | St. Louis

Kirin Brewery was founded in what country? | Japan

Pabst Brewing Company was headquartered in which U.S. city? | Milwaukee

Anheuser-Busch Brewery had its headquarters in which U.S. city? | St. Louis

Which beer is from the "land of sky blue waters"? | Hamm's

Which beer is advertised as "beechwood aged"? | Budweiser

Which beer is nicknamed "America's fire brewed beer"? | Stroh's

Miller Brewing Company featured what bird in its trademark? | Eagle

Kirin is a brewing company found mainly on what continent? | Asia

What country in the world is known for XXXX beer? | Australia

What is the most popular exported New Zealand beer? | Steinlager

From which country does Carlberg's beer originate? | Denmark

In what U.S. city is Weinhard Brewing Company located? | Portland, OR

Which beer claims to have "bottled beer taste in a can"? | Keystone

Where is Red Stripe beer brewed? | Kingston, Jamaica

Where is the Coors Brewing Company located? | Golden, Colorado

Which brand of beer has its brewery located in Latrobe, Pennsylvania? | Rolling Rock

How many cans are in a Lone Star "Texas 6-Pack"? | Eight

A gallon is equal to how many ounces? | 128 Ounces

Curacao is made from what dried peel? | Orange

What type of Schnapps is Ice 101? | Peppermint

Which soft drink is the oldest in the U.S.? | Dr. Pepper

What is Stilchester? | A Cheese

What might Italians call maize? | Polenta

From which country do french fries originate? | Belgium

In Peru, which color potatoes are grown, in addition to white? | Purple And Black

What are the two ingredients in a roux? | Flour And Fat (Butter)

What is also known as Liberty Cabbage? | Sauerkraut

What is kartofflen? | Potato Dumplings

What is the name of the flatbread eaten with most Indian cuisine? | Naan

What type of cuisine offers Dim Sum? | Chinese

What is Cioppino? | A Seafood Stew

What does Etoufee mean? | Smothered

What famous dish uses arborio rice? | Risotto

With which vegetable are Norwegian Lefse made? | Potatoes

What is the name of the bar where “Buffalo Wings” originated? | The Anchor Bar

What type of cheese is an ingredient in Tiramisu? | Mascarpone Cheese

What is a “sabayon”? | A Custard Dessert

What temperature should you not exceed when melting chocolate? | 120*F

What is the featured flavor in Mexican mole sauce? | Chocolate

What does the Italian term “Al Dente” mean in regards to pasta? | To The Teeth

What is the main flavoring agent in a Mornay Sauce? | Gruyere Cheese

What gives the drink known as a “Black Cow" its color? | Coffee

Trader Vic claims credit for creating what drink? | Mai Tai

What nationwide U.S. fast food chain opened the first

drive-in? | A&W

In 2005, how much beef and steak did the average American eat? | 67 Pounds

What is the first sign of the western zodiac? | Aries

What is the name of the currency used in Finland? | Markkaa