U N I V E R S I T Y OF T A R T U
Faculty of Mathematics and Computer Science
Institute of Computer Science

Riivo Talviste

Web-based data entry in privacy-preserving applications

Bachelor's Thesis (4 CP)

Supervisor: Dan Bogdanov, MSc

Author: ..... "....." June 2009
Supervisor: ..... "....." June 2009

Allowed to defence
Professor: ..... "....." June 2009

TARTU 2009

Contents

Introduction

In recent years the world wide web has grown more and more popular. Thus, many companies are also concentrating on providing their services on the web. Online surveys, questionaries, auctions have become very popular as they are both cheap and convenient for organizations and users. However, many of these services require users to input confidential data so the question of preserving the privacy of the data arises.

In this thesis we describe how this issue can be resolved at the moment and offer better solutions. We concentrate on the example of Denmark, where a secure nation-wide web-based auction was conducted in January 2008. We analyse the architecture of this auction and describe some of its constraints and shortcomings.

Our goal is to propose a privacy-preserving data collection architecture. It would enable us to gather and process confidential data so that the user inserting the data would be the only one who knows the input. Such a technology would make it easier to conduct web-based surveys, questionaries and auctions with confidential data.

In Section 1 of this paper we give a short overview of different legal aspects that regulate the handling of private data. We also briefly discuss the handling of confidential data in the current state of art. Section 2 lists and defines some of the cryptographic operations and protocols that are needed to understand this thesis. We also introduce the Sharemind framework which our proposed architectures use to perform private computations. In Section 3 we give an overall description of our proposed architecture, its security constraints and implementation details. Section 4 gives a more detailed overview of the JavaScript-based solution. The architecture described in this section is similar to the one used in the Danish auction system and thus has the same security risks. However, usability is improved by using the JavaScript technology. An architecture with stronger security guarantees is detailed in Section 5. This solution uses secure connections and is implemented with Adobe Flex framework.

1 Privacy-preserving data collection

1.1 Motivation

Data privacy issues can arise in response to information from a wide range of sources. In this paper we consider privacy as information privacy — the protection of personal data. The European Union directive on the protection of individuals with regard to the processing of personal data and on the free movement of such data [Cou95] defines personal data as any information relating to an identified or identifiable natural person (data subject), where an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity.

The Estonian Personal Data Protection Act [RTI07, RTI] defines sensitive personal data as:

1. data revealing political opinions or religious or philosophical beliefs, except data relating to being a member of a legal person in private law registered pursuant to the procedure provided by law;
2. data revealing ethnic or racial origin;
3. data on the state of health or disability;
4. data on genetic information;
5. biometric data (above all fingerprints, palm prints, eye iris images and genetic data);
6. information on sex life;
7. information on trade union membership;
8. information concerning commission of an offence or falling victim to an offence before a public court hearing, making of a decision in the matter of the offence or termination of the court proceeding in the matter.

Accessing information from these sources and many more is regulated by laws that make collecting, storing and analysing that kind of data extremely complicated. However, in many cases the parties that are interested in the data — data users — are in fact only interested in the data analysis results and not in the original data itself. Therefore, by showing data users only the end results of data processing, we could ensure the data donor's information privacy by guaranteeing that he or she is the only one who knows the complete input. To achieve that, we have to bring data entry as close to the data donor as possible, so that the confidential data would not pass through any third parties.

The Internet is a free and convenient medium for both data users and donors. Thus, online surveys and other web-based data-mining applications are gaining

more popularity. In this paper we analyse different methods to preserve data privacy in web-based applications.

1.2 State of the art

One way to preserve the data donor's privacy is to keep his or her identity secret. This means that no name, personal identification number or other identifying data is asked. Nevertheless, there are still cases [BJ06] where the identity of the data donor is revealed, based on the answers given by an anonymous data donor.

Another possible solution is to use secure connections between data donors and users. SSL (Secure Socket Layer) tunnels are one way to ensure that confidential data gets only to the right data user and that data integrity is preserved. However, the data user still sees the confidential data and could use it for selfish reasons or even distribute it to other parties.

In January 2008 in Denmark, a nation-wide exchange in a form of double auction was carried out to rearrange sugar beet growing contracts between local farmers and Danisco company. A survey was conducted together with the auction. It showed that farmers really care about the confidentiality of their bids. Thus, it was decided that the role of the auctioneer would be played by three independent parties using secure multiparty computation. To make bidding more convenient for farmers, a web-based application supporting secret sharing and secure multiparty computation was used. It was suggested to be the first large-scale and practical application of multiparty computation [BCD$^+$08, Des].

1.3 Our contribution

In this paper we analyse the architecture of the Danisco auction and suggest improvements from the point of security guarantees and usability. Firstly, we increase the usability of the Danisco auction's architecture by using JavaScript technology. Secondly, we propose a new architecture with improved security and implement it with Flex technology.

2 Preliminaries

2.1 Cryptographic operations

Symmetric-key cryptography

Definition Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$, respectively, where $\mathcal{K}$ is the key space. The encryption scheme is said to be *symmetric-key* if for each associated encryption/decryption key pair (e, d), it is computationally "easy" to determine d knowing only e, and to determine e from d [MvOV96, Chapter 1.5].

Symmetric-key encryption uses symmetric key algorithms, which means that one and the same key is used to both encrypt and decrypt messages. Hence, all of the communicating parties have to know that key. One of the major issues of symmetric-key encryption is the key distribution problem — before the parties can start to send encrypted messages, they have to agree upon and exchange the secret key that they will be using. At the moment, AES (Advanced Encryption Standard) is the standard used in symmetric-key encryption [DR01].

Public-key cryptography

Definition Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$, respectively. The encryption method is said to be a *public-key encryption scheme* if for each associated encryption/decryption pair (e, d), one key e (the *public key*) is made publicly available, while the other d (the *private key*) is kept secret. For the scheme to be *secure*, it must be computationally infeasible to compute d from e [MvOV96, Chapter 1.8].

In the case of public-key cryptography each party has its own pair of private and public keys. The sender of a secret message first encrypts it with the receiver's public key and then sends the encrypted message. The receiver can then decrypt the message with his private key. One of the advantages of public-key cryptography is that no initial exchange of secret keys is required. Public-key cryptography uses asymmetric key algorithms as the key used to encrypt the message is different from the key used to decrypt it. Currently, the standard algorithm used in public-key cryptography is RSA [RSA78].

Secret sharing

Definition Let s be the secret value. An algorithm $\mathbf{S}$ defines a *k-out-of-n threshold secret sharing scheme*, if it computes $\mathbf{S}(s) = [s_1, \ldots, s_n]$ and the following

conditions hold [Sha79, Dam02]:

1. **Correctness:** s is uniquely determined by any k shares from $\{s_1, \ldots, s_n\}$ and there exists an algorithm $\mathbf{S}'$ that efficiently computes s from these k shares.

2. **Privacy:** having access to any $k-1$ shares from $\{s_1, \ldots, s_n\}$ gives no information about the value of s, i.e., the probability distribution of $k-1$ shares is independent of s.

Secret sharing is mainly used to protect sensitive data like cryptographic keys. The data is split into a number of parts, called *shares*, that are distributed among separate parties who protect their shares. To reconstruct the original data, all the shares have to be recombined. The security assumptions of secret sharing schemes state that gaining access to a number of shares lower than the defined threshold gives no information about the secret value. The best-known secret sharing scheme was proposed by Shamir [Sha79].

In our case, secret sharing gives us a simple method to protect the privacy of the confidential data. The secret value is distributed into shares and each party participating in multiparty computation is given one of the shares. Hence, no single party can learn the original value.

2.2 Cryptographic protocols

Transport Layer Security

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols used in networks, mostly on the Internet. These protocols provide security and data integrity by encrypting network connections on the Transport Layer between its two endpoints.

TLS supports both *unilateral* (one way) and *bilateral* (both ways) authentication. However, in a common client-server scenario, only unilateral authentication is used — client authenticates the server by verifing its certificate. The client itself remains anonymous for the server. The bilateral authentication is more secure, but requires both the server and the client to have a valid certificate.

Secure multiparty computation

Secure multiparty computation (MPC) can be defined as a scenario with more than one party, where each party has an input value x_i and together they want to securely compute some function of these input values. Security in this case means guaranteeing a correct output of the function as well as preserving privacy of the

input values, even if some parties cheat. To achieve that, the parties must exchange messages and perform local computations to find the output value. Concretely, we assume that we have inputs $x_1, \ldots, x_n$, where party P_i knows x_i, and we want to compute $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ such that each party P_i will learn the value y_i, but nothing else [CD04].

The parties need communication channels in order to exchange messages. Two basic communication models are used in secure multiparty computation: *cryptographic model* and *information-theoretic model*. In the cryptographic model, the adversary has access to all the messages sent, however, it cannot modify the messages exchanged between honest nodes. In this case the communication is secure if the adversary cannot break the cryptographic primitive used on the channel. In the information-theoretic model all communication channels between the nodes are secured so the adversary gains no information about the messages exchanged between honest nodes. This stands even if the adversary has unbounded computing power [CD04].

The security model of a MPC protocol may be *universally composable*. This allows us to run several independent instances of the protocol sequentially or in parallel, without compromising the security [CD04].

2.3 The Sharemind framework

The Sharemind framework [BLW08] is a distributed virtual machine that allows us to process data without compromising its privacy. It is achieved by using secure multiparty computation protocols. The virtual machine consists of three data-mining servers, called *miners*. MPC protocols allow miners to run various computation algorithms on the data. However, none of the miners see the complete input data. The protocol suite of the Sharemind framework is based on an additive secret sharing scheme in the ring $\mathbb{Z}_{2^{32}}$.

Provided that we have a secret value s, it is secret shared as follows. We take two random values $s_1, s_2 \in \mathbb{Z}_{2^{32}}$ and compute $s_3 = (s - s_1 - s_2)\ mod\ 2^{32}$ so that $s_3 \in \mathbb{Z}_{2^{32}}$. Each of the three miners then gets one share from $\{s_1, s_2, s_3\}$. The secret value s can be reconstructed by computing $s = (s_1 + s_2 + s_3)\ mod\ 2^{32}$.

To achieve reasonable balance between security and efficiency, the Sharemind virtual machine works in the *semi-honest model*. In this model, all of the miners are assumed to follow the protocol, but they may also be curious about the data. So they could try to compute the secret value from the data available to them.

The Sharemind protocol currently supports privacy-preserving addition, multiplication with a constant, multiplication and comparison. These operations are sufficient for data analysis. These protocols are also universally composable, thus, they can be run sequentially to execute algorithms.

The current Sharemind implementation uses three data miners as it is the most optimal case. Two-party MPC is slow by construction, but adding parties increases the communication overhead. The privacy of the data is guaranteed if none of the miners shares its data with others. To satisfy these requirements one should choose three competing organizations as miners. The chosen organizations should have no incentive to collude with each other. Also they should be interested in keeping the privacy of the data. In most cases competing companies and data protection agencies are good choices.

The Sharemind framework has an API (Application Programming Interface) that can be used to interface it with a data analysis application. Applications built with the controller library can communicate with the miners, send data and ask computation results from them.

Each miner has its own database to save its shares. Thus it can be used to save the data and use it for later processing. Each miner also has a set of built-in algorithms and they give out only computation results.

3 A privacy-preserving data-collection system

3.1 Architecture

The lifecycle of private data in our proposed architecture is as follows (Figure 1). The confidential data produced in the client's computer is immediately distributed into shares. Each share is then sent to a different miner's web server that saves it in a local database. A daemon application periodically queries that database for new shares and converts and saves them to the corresponding miner's database. Miners can then use shares from their databases and MPC protocols to perform data analysis. The local database is used for buffering and helps to keep the architecture more robust. It means that collecting and secret sharing confidential data can be performed independently from processing the data.

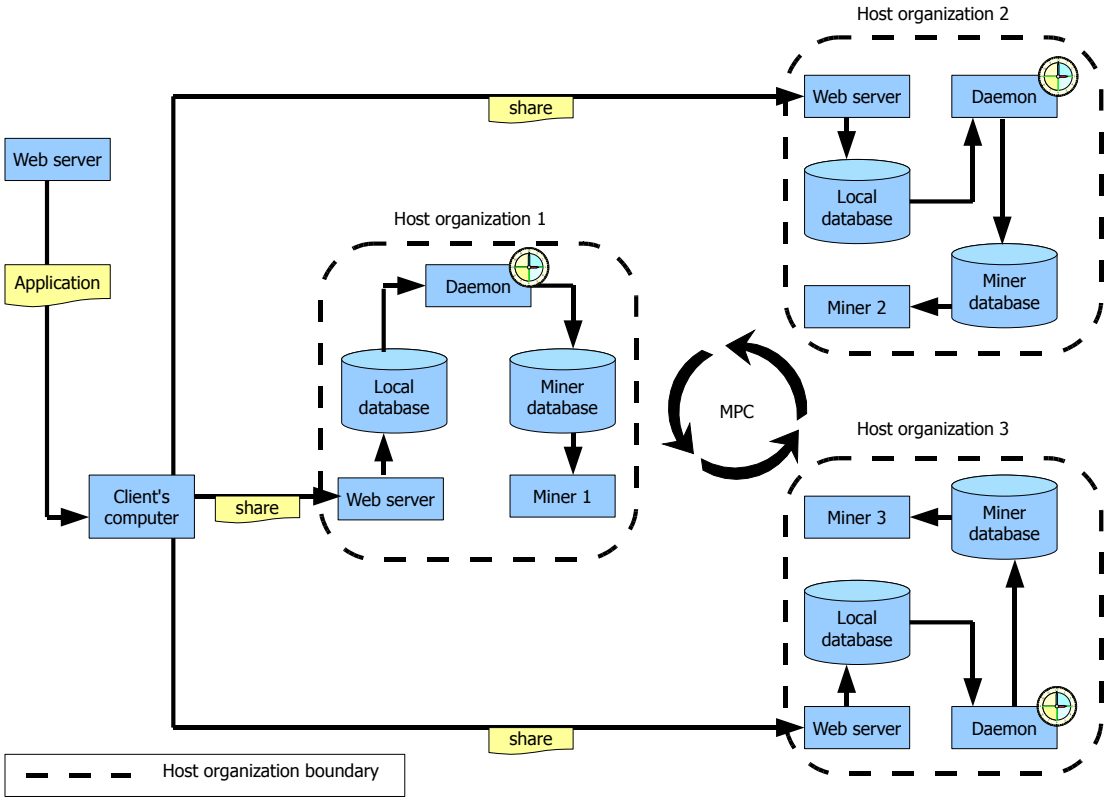

Figure 1: Data movement in the proposed architecture.

There is also another solution where the client sends its shares directly to all three miners. However, this would require miners to also act as web servers, possibly with SSL support. In the current implementation of the Sharemind framework the miners can only access shares from their own internal databases.

3.2 Security

In the following we describe how the security is preserved as the confidential data moves between components. If the connection from client's computer to a miner's web server is secured using the HTTPS protocol, then the TLS/SSL protocol provides the server authentication and channel security at that point. The solution with secure HTTPS connections between the client and a miner's web server is implemented with Flex technology and detailed in Section 5.

On the other hand, if the standard HTTP protocol is used, then the client also receives three public keys along with the application. Each of these keys is then used to encrypt one of the shares. Encrypted shares are then sent back to the web server that provided the application. The web server sends each encrypted share to the corresponding miner's web server that will then use its private key to decrypt the share. In this case public-key encryption is used for keeping the shares in secret. The case with unsecured HTTP connections is implemented with JavaScript technology and described in Section 4.

The data movement from the miner's web server to a local database and later to the miner's database takes place inside the organization. Thus, we rely on the security guarantees of that company or data protection agency. In the data processing phase where the miners use MPC protocols, the confidentiality of the data is provided by the Sharemind framework.

3.3 Implementation

To prove the feasibility of our solution, we implemented it with two technologies: JavaScript and Flex. These technologies have different features and constraints so the implemented architectures are also slightly different. The corresponding architectures are discussed in more details in Sections 4 and 5.

In the following, we explain how the proposed architecture works in a prototype survey application. Firstly, the user goes to a web page and receives a questionary as a part of a survey. In the case of JavaScript technology, the questionary is a HTML form, whereas in the case of Flex technology, it is Flex application embedded in the web page. The user can then fill in the questionary and click the "Send" button. Each of the inserted values is then distributed into shares and each share sent to a different miner's web server. Each miners's web server saves the received share to its local MySQL database. The web server also receives a unique session identifier from the user and saves it in the same database.

The WebControllerProxy is a daemon application which is scheduled to run periodically after a specific time interval. This application is built with the Sharemind framework controller library and used in each host organization. Its task is to poll the local MySQL database for new shares, sort them by the session identi-

fiers and save the shares to the miner's internal database. The miners can then use the shares from their internal databases and the MPC protocols of the Sharemind framework to run data analysis algorithms on the confidential data inserted by the user.

The described implementation is not a complete system. It is a proof-of-concept of the solutions proposed in this paper. The source code of the implementation is also included with the thesis.

4 A JavaScript-based solution

4.1 Technology overview

In this section we propose an architecture and its implementation with JavaScript, an ECMAScript dialect. JavaScript is a widely spread scripting language, built into most of the common web browsers like Mozilla Firefox, Apple Safari, Opera, Konqueror, Google Chrome, etc. Microsoft's Internet Explorer has its own dialect of ECMAScript, named JScript. However, it is compatible with JavaScript in most of the cases.

4.2 Solution architecture

The architecture itself is almost identical to the architecure used in the Danisco auction. The data collection scheme is as follows (Figure 2). The client connects to a web server hosting the data entry system and receives a JavaScript application that runs on the client's computer. Together with the application, the client receives three public keys of three different miner web servers. After the confidential data is entered, the application performs secret sharing and encrypts each share with a different public key. All of the encrypted shares are sent back to the initial web server, which distributes them among different miner web servers. Each miner's web server uses its corresponding private key to decrypt the share and save it in a local database. The rest of the process is the same as explained in the general case.

This architecture uses public-key encryption on the shares, because JavaScript applications cannot make direct secure connections to the miners' web servers. Most of the contemporary web browsers do not allow JavaScript applications to make connections outside the DNS domain where the application is hosted. This constraint is used in order to prevent *cross-site scripting* (XSS) with JavaScript. To solve that problem, the web server that hosts the JavaScript application is used as a proxy. However, to prevent that web server from recombining the shares, all of the shares are encrypted with different public key.

4.3 Security analysis

Since the architecture is the same as in the Danisco auction, the same security issues also apply. The web server that provides the client with JavaScript application and public keys, could send the client three self-generated keys instead. Thus, also having the matching private keys for these public keys, the owner of this web server could decrypt all the shares and therefore recombine the original data. This is possible, because all the encrypted shares are sent back to a single web server.

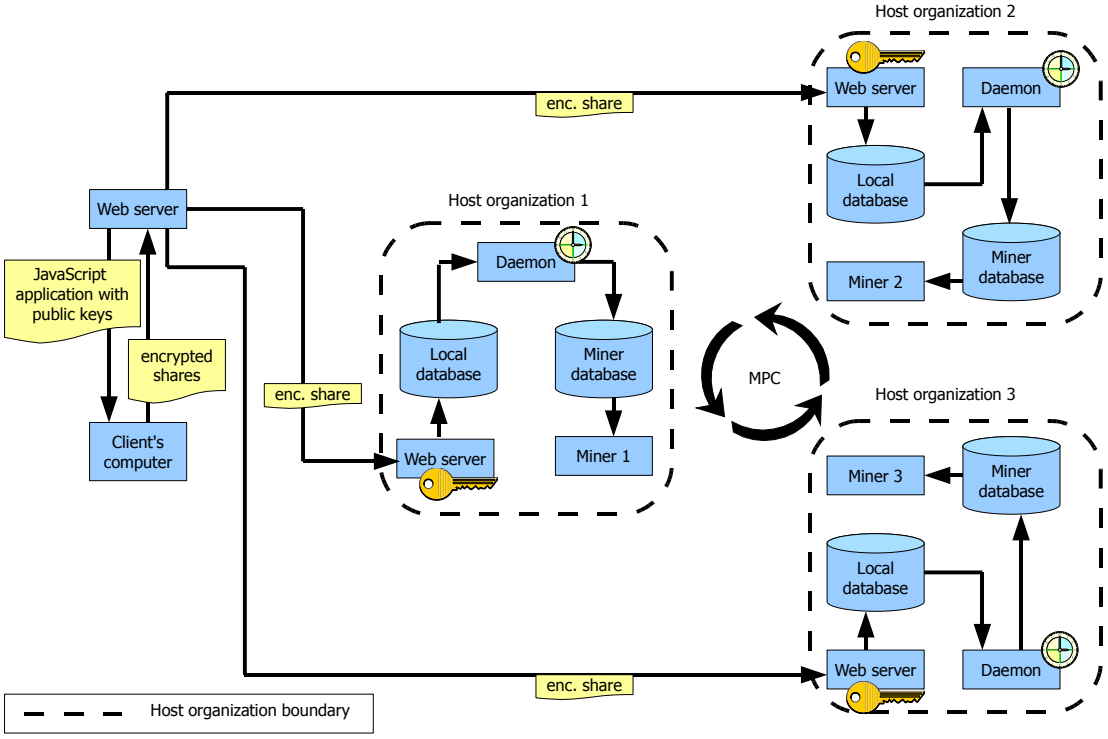

Figure 2: The proposed architecture using JavaScript technology and three shared public keys.

4.4 Implementation details

While the Danisco auction used a Java applet, we use JavaScript for client-side logic. The advantage is that most of the contemporary web browsers have a built-in JavaScript engine, which means that the client does not have to install any additional software. Moreover, JavaScript applications usually use fewer resources than Java applets which run in their own Java virtual machine.

The client-side application needs RSA encryption functions to perform encryption of secret shares. JavaScript does not have a built-in library for that, so in our implementation we use the RSA library for JavaScript by Tom Wu [Wu].

Moreover, JavaScript does not have a cryptographically secure pseudo-random number generator (CSPRNG) that would be sufficient to use in this architecture to perform secret sharing and public-key cryptography. Instead we use an Arcfour pseudo-random number generator (PRNG) and initialize it with the system time in milliseconds as a seed. The Arcfour PRNG is also included in the RSA library by Tom Wu. The absence of sufficient CSPRNG is a security risk that should be solved on the technology level.

5 A Flex-based solution

5.1 Technology overview

The architecture proposed in this section uses Adobe Flex as a technology platform. Flex is a free, open source framework for building web applications that deploy consistently on all major browsers, desktops, and operating systems. It provides a standards-based language and programming model that supports common design patterns. Flex uses MXML, a declarative XML-based language, to describe user interface layout and behaviors. ActionScript™ 3 is an object-oriented programming language, that is used in Flex to create client logic. Applications created with Flex can run in the browser using Adobe Flash® Player software or on the desktop on Adobe AIR™, the cross-operating system runtime [Incb]. In our implementation we use Flex version 3, the latest release of the technology at the time of completing this paper.

Flex is very similar to Adobe Flash technology. The main difference is that Flex is designed to be used by developers, whereas Flash is more for designers. They both produce a SWF file that is run in client's web browser by Adobe Flash Player software. In order to run the SWF file produced by Flex compiler, a client has to have Adobe Flash Player version 9 or newer installed. As a Millward Brown survey [Inca], conducted in December 2008 shows, about 99.0% of Internet-enabled desktops and wide range of devices are using Adobe Flash Player software platform, while Java is run by 81.0%. Thus, using Flex technology should not require any additional installations from most of the clients, which makes data-mining with this technology more convenient for end users.

It must be pointed out that while Flex is a free and open source framework, the Adobe Flash Player is a proprietary product of the Adobe Systems Incorporated. Also there are no free wide spread Flash players that could run SWF files generated by the Flex compiler.

5.2 Solution architecture

The solution is quite similar to the architecture discussed in previous section, with a few important dfferences. The data collecting process is detailed on Figure 3. The client connects to a web server and receives a client-side Flex application. After the confidential data is inserted by the client, the application performs secret sharing on it and connects to three different miner web servers, using secure HTTP (HTTPS) connections. Each miner's web server receives one of the shares and saves it in a local database. These shares are then accessed by a daemon application like described in the overall architecture in Section 3.1.

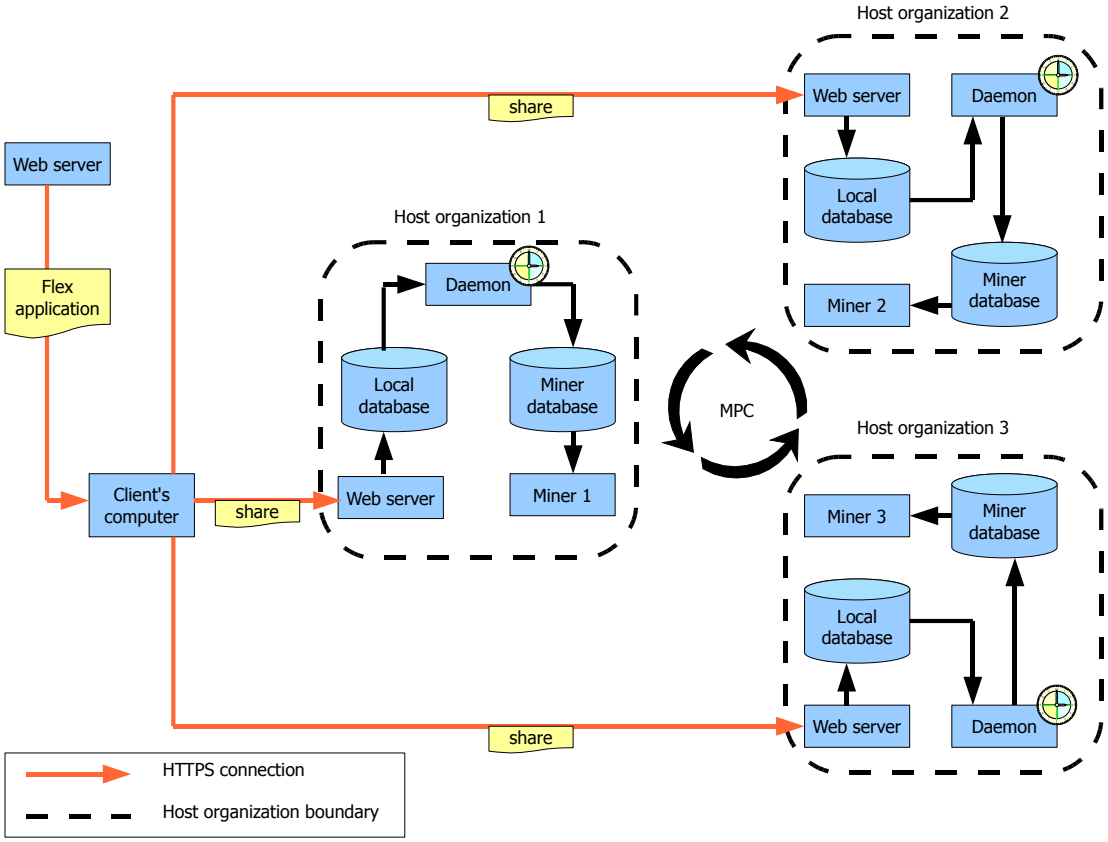

Figure 3: The proposed architecture with Flex technology and direct HTTPS connections.

5.3 Security analysis

In the architecture described in the previous section, the client has to trust the web server to give him or her the correct public keys. Fortunately, in this case, the client does not have to trust anyone unconditionally. The client connects to all the miners directly using the HTTPS protocol, which means that he or she can examine each miner's certificate and decide whether to trust them or not. Thus, we still use public-key cryptography, but the trust relationships are established directly.

In real life, however, the miners' certificates should be signed by a high level CA (Certificate Authority) that is already trusted by the client — this makes the process more transparent. Unfortunately, this also poses a security risk. The web server that sends the application to the client could send him or her an application with a modified set of miner addresses. Provided, that the miners have certificates that are already trusted by the client, the malicious owner of the web server would

receive all the shares and thus know the confidential data. However, finding a trusted CA to sign malicious certificates is not an easy task. This makes that kind of attack less probable.

5.4 Implementation details

As the data miners are controlled by different organizations or agencies, they are also most probably in different DNS domains. To allow Flex application to query servers on different domains than its own, the servers must have a XML file called `crossdomain.xml` in their document root directory. For example, provided that the client running the Flex application has an IP address 192.168.10.101, the contents of the `crossdomain.xml` should be similar to the following example:

```
<cross-domain-policy>
  <allow-access-from domain="192.168.10.101" secure="true"/>
</cross-domain-policy>
```

The `domain` attribute may also contain wildcards, e.g. `domain="*.example.com"` or `domain="*"`, to allow connections from different subdomains or from all IP addresses.

Since the Flex application uses HTTPS connections to access miner web servers, the Flex application itself also has to be served using secure HTTP connection. This behaviour can be overridden by setting the `secure` attribute in previous example to `false`, however, this creates an additional security risk.

As with the JavaScript language, Flex framework does not have a built-in CSPRNG to use in secret sharing the confidential data. To produce random numbers, we use the Park Miller "minimal standard" linear congruential pseudo-random number generator [PM88], which is implemented in Flex by Michael Baczynski [Bac].

This algorithm is capable of producing 31-bit random values, however, the Sharemind framework uses 32-bit values to hold secret data and shares. One possible solution to get 32-bit random values usable in cryptography is as follows. The Flex application makes a secure connection directly to each miner's web server and requests a random number. Since the web servers run on an operating system, they most probably have access to a CSPRNG. The Flex application then combines all three random numbers, e.g. XOR-s them together, and uses the result as the needed random value. This solution is secure if at least one web server is honest and does not log the random value that it sends to the application. However, a similar solution would not work in JavaScript, as the security contraints of web browsers do not allow JavaScript application to make HTTPS connections outside the application's DNS domain.

Conclusion

In this paper we look at the world wide web as a platform to gather data. Our main concern is how to process private data so that its confidentiality would be preserved. For example, this is the main challenge when conducting online surveys and auctions. At the moment anonymity and secure connections are commonly used to keep the data donor's identity and answers in private. However, this is not enough as the data donor still has to trust the party conducting the survey.

In January 2008, a large-scale auction was carried out in Denmark. The bids were kept confidential by the means of secret sharing and using three independent companies as auctioneer. The data processing phase used multiparty computation protocols, so no one learned the original data. In this paper we analyse the used architecture from the point of security and usability.

Firstly, we improve the usability of the architecture by using JavaScript technology on the client side. In the abovementioned auction, a Java applet was used as a client side applicaion to collect the data. However, a JavaScript interpreter is already built into most of the contemporary browsers, so no Java Runtime Environment is needed. Also, JavaScript applications require less resources. Unfortunately, with this architecture, the client needs to trust the web server that hosts the survey.

Secondly, we propose a new architecture that eliminates a security risk by using secure HTTP connections. This eliminates the requirement for encrypting shares as TLS/SSL provides the necessary security. We chose Adobe Flex platform to implement our proposed architecture. Flex is a free open source framework designed to develop rich client-side internet applications. Applications written in Flex are run by Adobe Flash Player. Since most of the internet-enabled computers already have Flash Player installed, applications built with Flex are convenient for the clients.

We implemented both the JavaScript-based and the Flex-based solutions as a part of this thesis. In our case, we used the Sharemind framework as the backend privacy-preserving computation engine.

Both presented solutions can be used for private web-based data collection either for the Sharemind framework or other secure multiparty computation systems.

Veebipõhine andmesisestus kõrgete privaatsusnõuetega rakendustes

Bakalaureusetöö (4 AP)

Riivo Talviste

Resümee

Veebi kiire areng on endaga kaasa toonud mitmete teenuste muutumise veebipõhiseks. Viimasel ajal on üha populaarsemaks saanud mitmesugused veebipõhised ankeedid, küsitlused ja oksjonid. Mitmed sellised rakendused eeldavad kasutajalt konfidentsiaalsete andmete sisestust, kuid samas on delikaatsete andmete töötlemine reguleeritud nii Euroopa Liidu kui ka Eesti seadustega. Seega tekib küsimus, kuidas töödelda kasutaja sisestatud andmeid nii, et nende privaatsus säiliks.

Enamasti kasutatakse andmete privaatsuse säilitamiseks kliendi anonüümsust või turvalisi suhtluskanaleid. Sellest aga tihti ei piisa, kuna on mitmeid juhtumeid, kus ananüümne klient on tuvastatud ainuüksi tema poolt antud vastuste põhjal. Samas, turvaliste suhtluskanalite kasutamise korral peab klient siiski usaldama organisatsiooni, kellele ta oma andmeid jagab.

Taanis viidi 2008. aasta jaanuaris eksperimendi korras läbi suureulatuslik oksjon. Kasutajate panuste salajas hoidmiseks kasutati andmete ühissalastust ning oksjonipidaja rolli täitsid kolm sõltumatut organisatsiooni, kes omavahel koostööd ei teinud. Andmete töötlemiseks kasutasid osapooled turvalise ühisarvutuse protokolle. Käesolevas töös analüüsime ja täiendame kasutatud süsteemi turvalisust ja kasutatavust.

Esmalt parandame me Taanis kasutatud süsteemi mugavust, realiseerides selle sama arhitektuuri JavaScript tehnoloogia abil. Taanis toimunud oksjonil kasutati Java rakendit, aga see nõuab kasutajalt lisatarkvara olemasolu. Samas JavaScripti interpretaator on enamikesse kaasaegsetesse veebilehitsejatesse juba sisse ehitatud ning JavaScript nõuab arvutilt ka vähem ressursse kui Java virtuaalmasin. Kahjuks ei lahenda JavaScripti kasutuselevõtt aga selle arhitektuuri turvariske — kasutaja peab ikkagi täielikult usaldama veebirakenduse tegijat, et see tema andmeid ei kuritarvitaks.

Järgmisena eemaldame me selle turvariski, kasutades turvalisi otseühendusi kliendi ja andmetöötlust läbi viivate osapoolte vahel. Sellest tulenevalt ei pea kliendi arvutis enam ühissalastatud andmeid krüpteerima, kuna vajaliku turvalisuse tagab turvaline sidekanal. Samuti ei pea kõiki osakuid saatma enam ühele osapoolele, sest iga osapoolega on loodud eraldi usaldatav ühendus. Selle lahenduse realiseerimiseks valisime Adobe Flex platvormi. Flex on tasuta avatud lähtekoodiga

raamistik, millega on võimalik luua interaktiivseid veebirakendusi, mis jooksevad kliendi veebilehitsejas. Flex rakendused nõuavad kliendi arvutis Adobe Flash Playeri olemasolu. Kuna viimane on aga enamikes arvutites juba olemas, on selle platvormi kasutamine kasutajale mugav.

Kõik selles töös kirjeldatud arhitektuurid sobivad veebipõhiseks privaatsust säilitavaks andmekogumiseks ning on kasutatavad ühisarvutuse keskkonnas. Meie välja pakutud arhitektuurid kasutavad turvalise ühisarvutuse teostamiseks Sharemindi raamistikku.

References

[Bac] Michael Baczynski. A good pseudo-random number generator (prng). Published online at `http://lab.polygonal.de/2007/04/21/a-good-pseudo-random-number-generator-prng/`. Last visited on May 26, 2009.

[BCD+08] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael Schwartzbach, and Tomas Toft. Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068, 2008. `http://eprint.iacr.org/`.

[BJ06] Michael Barbaro and Tom Zeller Jr. A face is exposed for AOL searcher no. 4417749. The New York Times, August 9th, 2006.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2008.

[CD04] Ronald Cramer and Ivan Damgård. Multiparty computation, an introduction. Course Notes, 2004.

[Cou95] European Council. Directive 95/46 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. OJ L 281, 1995.

[Dam02] Ivan Damgård. Secret sharing. Course notes, 2002.

[Des] Partisia Market Design. Partisia market design. Published online at `http://partisia.com/`. Last visited on May 26, 2009.

[DR01] Joan Daemen and Vincent Rijmen. Advanced encryption standard (aes) (fips 197). Technical report, Katholijke Universiteit Leuven/ESAT, 2001.

[Inca] Adobe Systems Incorporated. Flash player penetration. Published online at `http://www.adobe.com/products/player_census/flashplayer/`. Last visited on April 23, 2009.

[Incb] Adobe Systems Incorporated. Flex overview. Published online at `http://www.adobe.com/products/flex/overview/`. Last visited on April 23, 2009.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[PM88] Stephen K. Park and Keith W. Miller. Random number generators: Good ones are hard to find. *Commun. ACM*, 31(10):1192–1201, 1988.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[RTI] Personal data protection act. RTI, 16.03.2007, 24, 127; Published online at `http://www.legaltext.ee/text/en/XXXX041.htm`. Last visited on May 29, 2009.

[RTI07] Isikuandmete kaitse seadus. 15.02.2007. - RT I 2007, 24, 127; RT I 2007, 68, 421, 2007.

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

[Wu] Tom Wu. Bigintegers and rsa in javascript. Published online at `http://www-cs-students.stanford.edu/~tjw/jsbn/`. Last visited on May 26, 2009.