CYBERNETICA
Institute of Information Security

An improved method for privacy-preserving web-based data collection

Riivo Talviste, Dan Bogdanov

T-4-5 / 2009

Riivo Talviste[1], Dan Bogdanov[1,2].
[1] University of Tartu, Institute of Computer Science
[2] AS Cybernetica, Institute of Information Security

The research reported here was supported by:

1. Estonian Science foundation, grant(s) No. 8058,

2. the target funded theme SF0012708s06 "Theoretical and Practical Security of Heterogenous Information Systems",

3. the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, and the Software Technology and Applications Competence Centre, STACC,

4. EU FP6-IST project AEOLUS (contract no. IST-15964).

Cybernetica research reports are available online at
`http://research.cyber.ee/`

Mailing address:
AS Cybernetica
Akadeemia tee 21
12618 Tallinn
Estonia

An improved method for privacy-preserving web-based data collection

Riivo Talviste, Dan Bogdanov

January 25, 2010

1 Introduction

Lately, people have become more concerned about their privacy. Also processing private data is regulated by laws. This means that confidentiality is an important issue in many (computer) applications. For example, e-voting is an information system with multiple parties, who have something to hide from each other — everybody's vote has to be kept in secret.

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. Together with the auction, a survey was conducted, showing that farmers really care about the confidentiality of their bids (Figure 1). Thus, it was decided that the role of the auctioneer would be played by three independent parties using secure multiparty computation (MPC). To make bidding more convenient for farmers, a web-based application was used. It was the first large-scale and practical application of multiparty computation [BCD+08].

In this paper 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.

Section 2 of this paper describes the forementioned system in more detail and summarizes some of its constraints and shortcomings. In Section 3 we give an overall description of our proposed improved 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

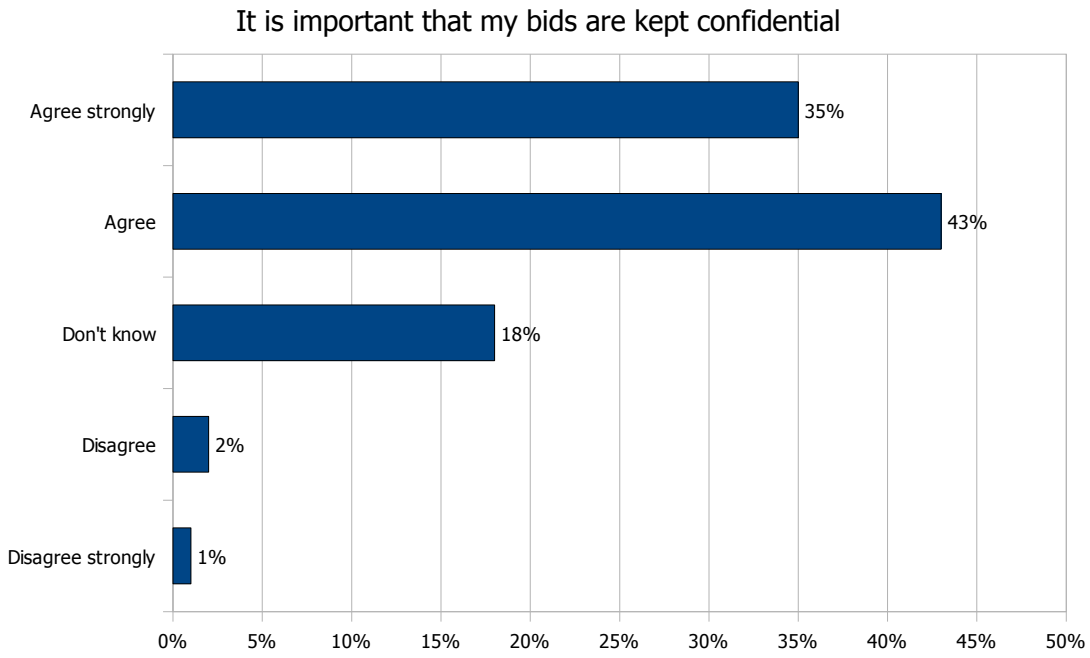

Figure 1: Survey results showing farmers' confidentiality expectations.

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.

2 State of the art

2.1 The Danisco auction

In Denmark several thousand farmers produce sugar beets, which they sell to the Danisco company — the only sugar beets processor on the local market. Farmers have contracts that give them rights and obligation to deliver a certain amount of sugar beets to Danisco who pays them, according to a pricing scheme given in their contracts. With recent changes in EU funding schema, however, these contracts had to be reallocated to farmers where the production pays off best. It was decided that this must be done via nation-wide double auction [BCD+08].

Double auction works in a way that each seller declares how much is he or she

willing to sell for a given price range and also each buyer specifies how much is he or she willing to buy for each price range. From these bids, auctioneer calculates the *market clearing price*. All sellers who accepted to sell for that price or lower, get to sell their goods at that price and all buyers who offered more than that price, get to buy the goods.

However, like the forementioned survey showed, farmers wanted their bids to be kept in secret, as their bids would clearly reveal information about their economic position and productivity. Thus, they would have been reluctant to use the Danisco company as auctioneer. On the other hand, using an independent third party as an auctioneer was unacceptable for Danisco as the contracts contain vital information for the company. Hence, it was decided that the role of auctioneer would be played by a group of three independent parties: the Danisco company, DKS (a sugar beet growers association) and SIMAP (Secure Information Management and Processing) project, which has been responsible for the practical application of MPC, described in [BCD+08].

The auction itself was held in two phases (Figure 2). In the first phase each farmer logged in to his or her existing account on Danisco website. The initiated session was forwarded to another web server, which provided farmer with a Java applet. Together with this applet the farmer received three public keys, each belonging to one of the auctioneers. Farmer placed his or her bid, which was split into three pieces using secret sharing. Each piece was then encrypted with different public key and sent back to the web server which stored them in a local database.

In the second phase each auctioneering party sent a representative who copied their shares from the web server database and used their matching private keys to decrypt them. The market clearing price was then calculated using decrypted shares from the representatives and multiparty computation protocols.

2.2 Problem statement

The mentioned system worked as expected. However, it has a major security risk — the web server that provides the farmer with Java applet and three private keys, could send him or her three self-generated public 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 know all the farmers' bids. This is possible, because encrypted shares are sent back to a single web server.

Moreover, Java applets require a significant amount of resources and Java Runtime Environment (JRE) is not installed in every client computer. This makes conducting large-scale web-based surveys or auctions with suggested architecture more

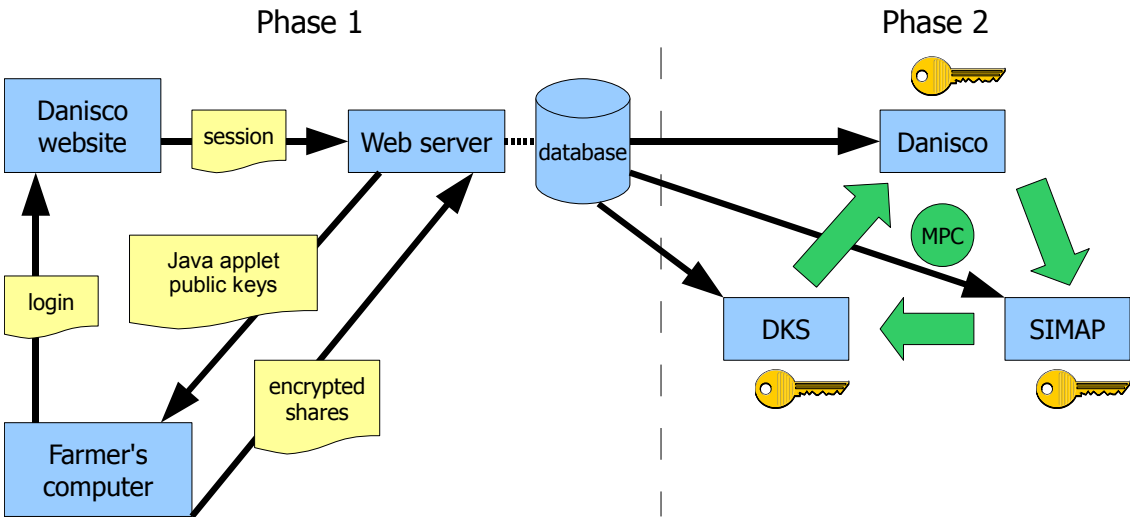

Figure 2: The architecture used in the Danisco auction.

complicated.

The main contribution of this paper is to propose an architecture that eliminates this security risk and try a different client-side technology that is easier for the client to use.

3 Web-based data gathering with improved confidentiality

3.1 Architecture

Our solution is quite similar to the architecture discussed in previous section, with a few important differences. We use the Sharemind framework [BLW08, BK09] as our backend system to store and later process the data. Data-analysis, however, is not in the scope of this paper. Our intent is to gather and save the data to the data storage and processing units, which in the Sharemind framework are called *miners*.

The lifecycle of private data in our proposed architecture is as follows (Figure 3). 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.

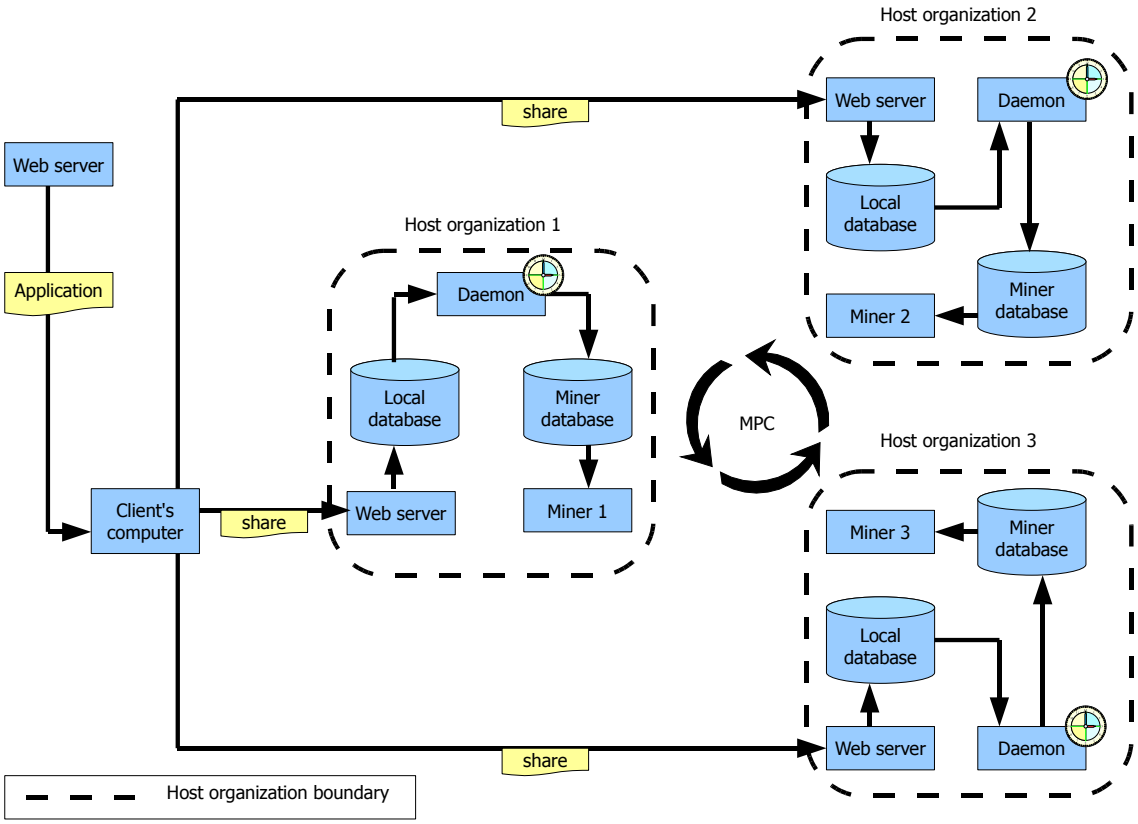

Figure 3: Data movement in the proposed architecture.

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 identifiers 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.

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 4). 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.

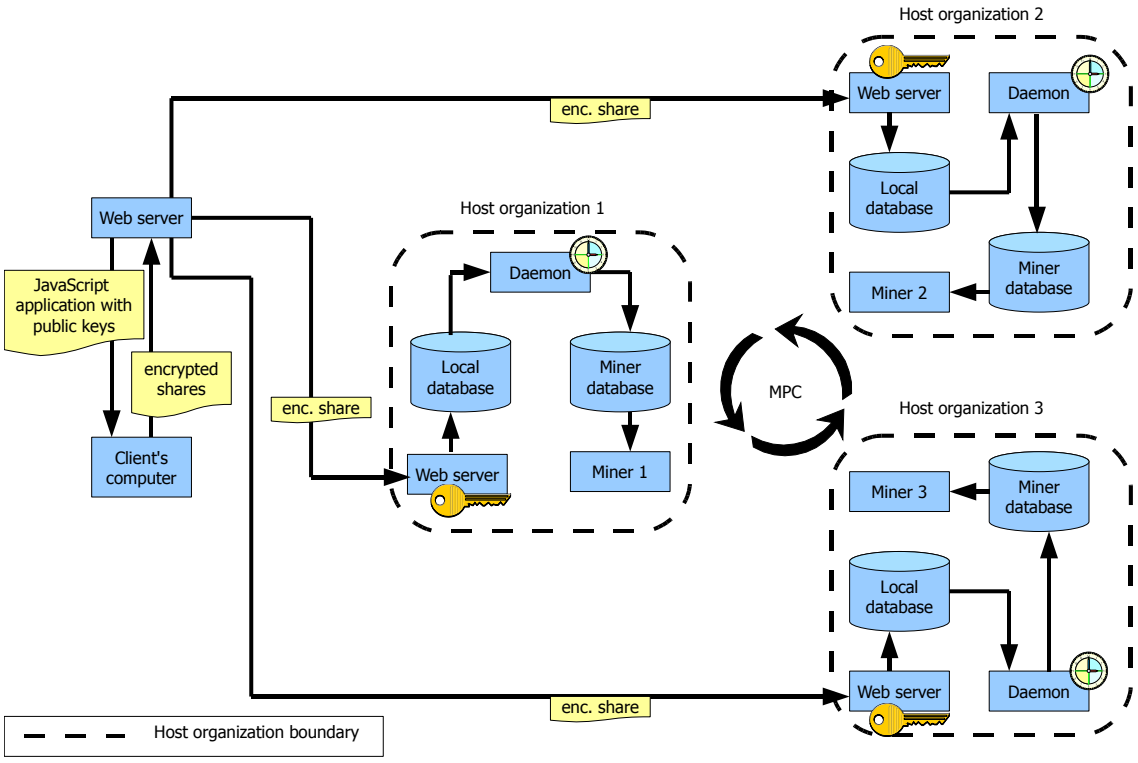

Figure 4: The proposed architecture using JavaScript technology and three shared public keys.

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.

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 open source 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 5. 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.

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

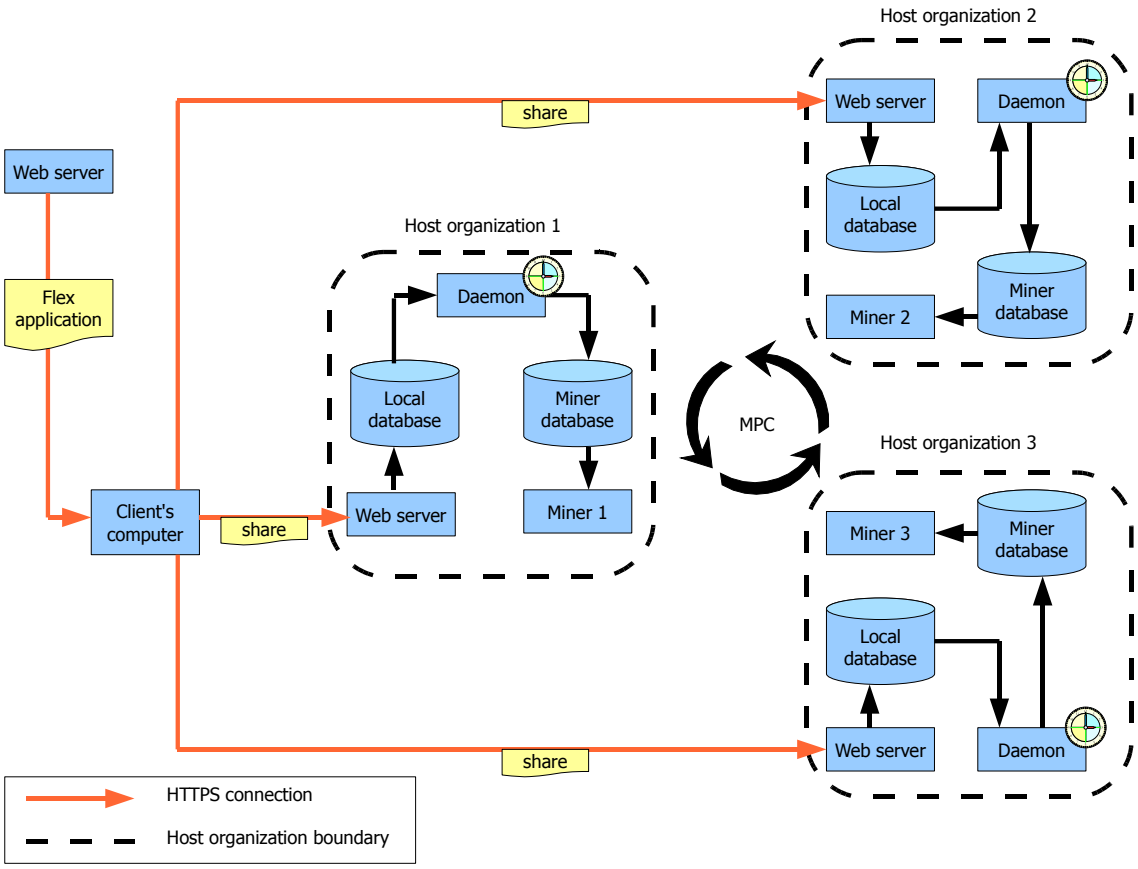

Figure 5: The proposed architecture with Flex technology and direct HTTPS connections.

`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.

6 Conclusion

First large-scale practical applications of multiparty computation are already in use. 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 paper. 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.

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/`.

[BK09] Dan Bogdanov and Liina Kamm. Architectures for privacy-preserving information systems. to appear, 2009.

[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.

[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.

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

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