Online Elections in Practice: Improvement and Application of the Helios Voting System

Pieter Maene

***Abstract*—Helios is a voter-verifiable online voting system that supports threshold encryption. The procedure to manage an election is discussed here. The interface of the system was modified to improve the usability. It was then successfully used in a real election. The Web Cryptography API adds cryptographic functionality to user agents. Its performance is compared to existing JavaScript crypto libraries. The results are promising, especially for more demanding calculations. The largest disadvantages are the limited set of supported algorithms and lack of low-level functions. This makes it hard to develop custom encryption schemes.**

I. Introduction

Elections are an important part of our democratic society. In 2014, national elections will be held in 40 countries and 42% of the world population will be able to cast its vote.[1] It is therefore necessary to have trustworthy voting system. In current systems, voters usually have to trust the election organiser. Voter-verifiable systems allow them to verify whether their ballot was registered correctly and check the election result. Helios is such a voter-verifiable system, that can be used for online elections. It is an open-source project developed by Ben Adida.[2] Online elections can only be used in situations where coercion is not a great threat, though. Robbert Coeckelbergh added threshold encryption to it, so that one trustee can no longer stall the election.[3] This paper will gave the procedure that should be followed to create an election in Helios. The interface of the system was also modified to better support this procedure. Finally, the system was applied in a real election.

The Web Cryptography API is a new W3C specification that will add cryptographic functionality to user agents.[4] The performance of this new API will be compared to that of existing JavaScript crypto libraries. If better, it could improve the usability of web applications that need user-level cryptography.

II. Procedure

A. Preparation

The trustees will be responsible to decrypt the result. It is therefore important to give some thought to who this role will be given. If threshold encryption is begin used, the election administrator should also determine the threshold scheme.

For each question the number of answers that can be selected by the voter can be specified. The result for each answer can be shown either as the absolute number of votes it received, or its relative position with respect to the other answers.

Helios supports both open and closed elections. Anybody can vote in the former, while only specific voters are allowed to cast their ballot in the latter.

B. Helios

The first step is the creation of the election. Here, all basic information like the name should be given. It is also possible to enable options here, like threshold encryption or whether the election is private or not. The start and end date of the election can be controlled here as well.

After creating the election, the trustees have to be added. This step was placed first in the procedure because it takes the longest to complete. After being added, the trustee receives an e-mail with the link to his trustee dashboard.

If threshold encryption is not being used, each trustee simply has to generate a key pair. Once all trustees have done this, the election administrator can freeze the election. If threshold encryption was enabled, the key ceremony becomes more complicated. After adding all trustees, the election administrator can define the threshold scheme. The trustees can then start the following ceremony.

1) The trustee has to generate his communication keys and upload the public keys to the server. They will be used to encrypt and sign the communication between them.
2) Once all trustees have done this, each of them can generate his encrypted shares and send them to the others.
3) After receiving a share from all other trustees, the trustee can decrypt and add them to obtain his share of the key that will be used to encrypt the ballots. It will serve as his private key for the election and he has to upload the corresponding public key.

When all trustees have done this, the election can be frozen. Since the trustees will have to wait for the others. Therefore, e-mails are sent to all of them when action is required.

While waiting for the trustees to complete the key ceremony, the election administrator can continue setting up the election. He can enter the questions and upload the voter lists if the election is private.

Once the election is frozen, voters will be able to cast their ballot unless a specific starting time was entered. If no end time was given, the election will be closed at the administrator's discretion.

After the election has been closed, the homomorphic tally can be decrypted. If threshold encryption is not enabled, all trustees will have to calculate their decryption factor and submit it. Otherwise, only the number of trustees defined in the threshold scheme will have to do this.

III. Interface

Most of our work on the interface to improve the usability concentrated on two big parts of Helios. First, the admin was modified to better reflect the procedure discussed in Section II. Second, the trustee dashboard was also simplified. Additionally, the layout was streamlined throughout the application.

A. Admin

The old interface integrated the administrative functions in the election view. This made it hard for the election admin to keep track of his progress in the procedure. It was therefore decided to move this functionality to a separate page. It shows the entire procedure and clearly indicates which steps have been completed. Since the actions lead away from this page, a bar was put on top of all pages where the progress is indicated as well.

The workflow for an election with threshold encryption was thoroughly changed. Originally, a public bulletin board was available where anybody could upload a pair of communication keys. The admin could only add trustees to an election from a list of people who had done this. This meant that he had to wait for all trustees to do this before he could define the threshold scheme and continue setting up the election. This required a secondary path of communication, outside of Helios. Because the bulletin board was publicly accessible, an attacker could impersonate a trustee. Unless he has access to the trustee's e-mail account, he would not receive the URL to the trustee dashboard. In that case, he would have been able to stall the key ceremony.

The bulletin board was removed completely and trustees are immediately added to the election. They then receive an e-mail with a unique link to their trustee dashboard where they can generate their communication keys.

B. Trustee Dashboard

The trustees will perform all actions of the key ceremony in the trustee dashboard. The trustee's role is explained and an overview of all steps in the ceremony is shown in the same way as the election procedure on the admin page.

The trustee will generate three key pairs during the ceremony. The private keys are stored locally in JSON files. Originally, the JSON was simply shown to the user and he had to manually save it to a file. When the key was needed, the file's contents needed to be copied again to a text area.

The HTML5 File API offers two API interfaces that can be used to ease this process.[5] The first is the `Blob` interface that can be used to generate files in JavaScript. They can then be offered as a download in the browser. The second is the `FileReader` interface which allows a web developer to read files that conform to the `Blob` or `File` interfaces.

They are used in the trustee dashboard to offer the secret keys as a download. The details of the key are now hidden from the trustee. When his key is needed, he has to select this file in an input element. The file's contents are then read to the text area. These adjustments make it a lot easier for the trustee to manage his keys.

In addition to using this API, the two secret keys of the communication pair are stored in a single file. Since the distinction between two was only confusing to the trustee, this simplifies the process.

IV. Application

V. Web Cryptography API

Currently, web developers implement cryptographic functionality in JavaScript. Despite great improvements in recent years, it is still slower than native code.[6][7][8] In 2012 a working group was started by W3C to write the Web Cryptography API specification, which would add cryptographic functionality to user agents.[9] The specification defines the `SubtleCrypto` interface which has several high-level functions.[4] There are both methods to manage the keys and to encrypt data. The API only supports a specific set of algorithms, nor does each algorithm support all functions of the interface.

Since the specification has not been completed yet, most user agents do not support it. Internet Explorer 11 is the only browser that already has an implementation, but it is missing some algorithms.[10] Therefore, the NfWebCrypto polyfill was used for the benchmarks. This is a C++ plugin for Chrome, so its performance should be comparable to a real implementation.

A. Modular Exponentation

Helios uses ElGamal to encrypt the ballots, which is an algorithm not supported by the API. The modular exponentiations are responsible for the majority of the calculation time. An improvement there would impact overall performance.

Since the API only offers high-level functions, this had to implemented separately. Diffie-Hellman is one of the supported algorithms. The `deriveKey` method uses Eq. 1 to calculate the shared secret.[11]

$$K = (g^a)^b \mod p \tag{1}$$

In this equation g^a is the public key of A and b the private key of B. If the public and private key can be specified, it can be used to calculate the modular exponentiation $x^y \mod p$. It is possible to import a specific private key into the user agent's key store. For the Diffie-Hellman algorithm, the specification only allows private keys in the PCKS8 format to be imported. To make testing easier, the NfWebCrypto plugin was modified to allow the import of raw keys. After calculating the shared secret, the result can be obtained with the `exportKey` function.

The performance of the NfWebCrypto library is compared to that of JSBN and Leemon, two JavaScript big number libraries.[12][13] A 1,024-bit exponent is used, which is the size of the keys used by Helios. The calculation is repeated 1,000 times for each library. The results are shown in Fig. 1 and Table I. The first exponentiation of the NfWebCrypto plugin took a lot longer, which explains the large variance. The NfWebCrypto plugin result was not correct. The communication with the plugin has to be `base64` encoded, which resulted in conversion issues between C++ and JavaScript.

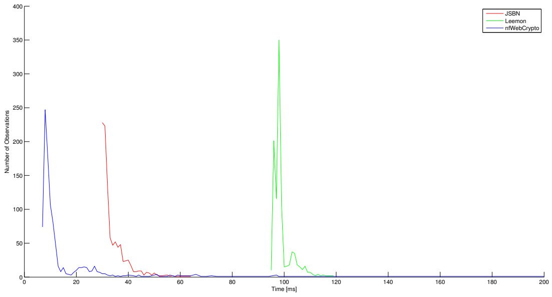

Fig. 1. Modular Exponentiation

Library	Gemiddelde [ms]	Variantie [ms]
JSBN	33,9250	26,5019
Leemon	99,1470	15,0785
NfWebCrypto	16,0670	470,6251

TABLE I
MODULAR EXPONENTIATION

B. RSA

RSA was used to make a second comparison between JSBN and NfWebCrypto. This cipher isn't used by Helios, but the results are very interesting. A standard value of 65,537 was used for the public exponent of the encryption. Notice that this is a lot shorter than the exponents used in Section V-A. The modulus is 1,024 bits large.

JSBN has specific functions for RSA encryption and decryption. The Web Cryptography API supports the `RSAES-PKCS1-v1_5` algorithm.[14] Although public keys in the SPKI format can be imported, this proved difficult. Therefore, a new key was generated for each encryption. The time required for this is not taken into account. The results are shown in Fig. 2 and Table II. Surprisingly, the JavaScript implementation is faster than the NfWebCrypto plugin. This is caused by communication overhead.

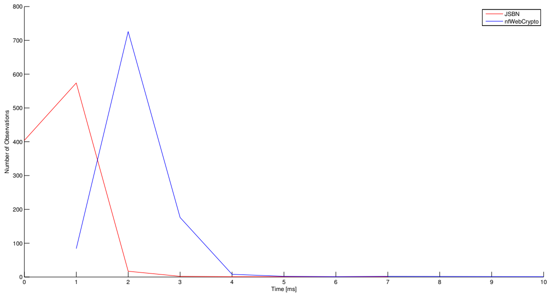

Fig. 2. RSA

Library	Gemiddelde [ms]	Variantie [ms]
JSBN	0,6310	0,3632
NfWebCrypto	2,1360	0,4219

TABLE II
RSA

C. Conclusion

The Web Cryptography API will give developers access to fast cryptographic functions in the browser. The small set of supported algorithms limits the possible applications. It also proved to be hard to import existing keys.

VI. CONCLUSION

REFERENCES

[1] T. Economist, "The 2014 ballot boxes," http://www.economist.com/node/21592755, 2014, [Online; Accessed 17-May-2014].

[2] B. Adida, "Helios Documentation," http://documentation.heliosvoting.org, 2014, [Online; Accessed 8-May-2014].

[3] R. Coeckelbergh, "Application and axtension of the Helios online voting system," 2013.

[4] R. Sleevi and M. Watson, "Web Cryptography API," W3C, W3C Last Call Working Draft 12 September 2013, 2014. [Online]. Available: http://www.w3.org/TR/2013/WD-FileAPI-20130912/

[5] A. Ranganathan and J. Sicking, "File API," http://www.w3.org/TR/2014/WD-WebCryptoAPI-20140325/, W3C, W3C Last Call Working Draft 25 March 2014, 2014.

[6] J. Resig, "JavaScript Performance Rundown," http://ejohn.org/blog/javascript-performance-rundown/, 2008, [Online; Accessed 5-May-2014].

[7] C. Cois, "JavaScript Performance Rundown, 2012," http://codehenge.net/blog/2012/08/javascript-performance-rundown-2012/, 2012, [Online; Accessed 5-May-2014].

[8] F. Smedberg, "Performance Analysis of JavaScript," 2010.

[9] W3C, "Web Cryptography Working Group Wiki," http://www.w3.org/2012/webcrypto/wiki/index.php?title=Main_Page&oldid=483, 2014, [Online; Accessed 5-May-2014].

[10] Microsoft, "Web Cryptography," http://msdn.microsoft.com/en-us/library/ie/dn302338(v=vs.85).aspx, 2014, [Online; Accessed 6-May-2014].

[11] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[12] T. Wu, "RSA and ECC in JavaScript," http://www-cs-students.stanford.edu/~tjw/jsbn/, 2009, [Online; Accessed 6-May-2014].

[13] L. Baird, "Big Integers in JavaScript," http://leemon.com/crypto/BigInt.html, 2009, [Online; Accessed 6-May-2014].

[14] J. Jonsson and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1," http://www.ietf.org/rfc/rfc3447.txt, Internet Engineering Task Force, 2003.

Pieter Maene