Two-factor Authentication and Digital Signing for an Enterprise System utilizing Yubikey

David Hilm
David Rahim

Supervisor: Jody Foo
Examiner: Peter Dalenius
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår. Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© David Hilm, David Rahim
Two-factor Authentication and Digital Signing for an Enterprise System utilizing Yubikey

David Hilm  
Student, LiU  
Linköping, Sweden  
Davhi440@student.liu.se

David Rahim  
Student, LiU  
Linköping, Sweden  
Davra786@student.liu.se

ABSTRACT
The use of a second factor to increase the security of systems is growing and has continued to do so for a long time. This thesis explores options for implementation to use a YubiKey as an authentication method (OTP) as well as for signing digital transactions through a web browser client. Measures of network overhead that occurs in conjunction with Digital Signing of transactions are also disclosed. Our findings show that YubiKey provides flexible and readily available solutions that can be used with only small implementations for OTP authentication. It is also shown that the major concern for implementing a solution for a web browser is to intuitively use certificates stored on a USB-device without installing any plugins or with the use of a third-party application running on the client machine.

Author Keywords
Digital signature; Security; Two-factor Authentication; Authentication; Cryptography; YubiKey; JavaScript; WebCrypto; Universal Second Factor;

INTRODUCTION
The demand for safer ways to authenticate users has become a vital part of having a good and secure environment on most platforms today. This is true both for the private sector as well as for enterprises. Not only in regards to verify the identity of users, but also in more and more cases it is expected to be able to use it as a digital signature for confirming transactions or signing documents. Two-factor authentication is quickly becoming the standard procedure for a lot of services.

Some of the biggest banks in Sweden has by a joint project developed a tool for authentication called BankID [1]. This tool has become essential for citizens in Sweden as a part of daily life and is used for banking, services provided by the government as well as other public services [1]. BankID can be used by storing a file locally on your device, such as your personal computer, on a smart card or, in the most common way, on your mobile phone.

There are some limitations in using the BankID service, such as it is only viable for Swedish services that uses it as a method for identification. With this limitation in mind, other services needs to be explored, as we need to find ways to incorporate markets outside of Sweden as well as a more developer friendly platform. The aspiration is however to achieve a similar functionality where you can use the service as an identification method as well as a digital signature.

In order to securely create a digital signature, a cryptographic model needs to be applied [2]. This will ensure that a transaction or document that has been signed by a user cannot be modified without it being detected. If a transaction is tampered with after it has been signed off by a user, it will no longer be verifiable – thus making it invalid.

Motivation
By increasing the level of security of a system, we also wish to expand the awareness of this issue to its users. To find solutions that is beneficial to all parties included is what drives the exploration and motivates further development within this area. We hope to encourage research by identifying certain areas that is halting developers from implementing solutions similar to the one we propose.

Purpose
Our main goal will be to find a solution that is applicable for large scale business platforms, where a typical user case would be where an employee needs to confirm a transaction. These transaction could also be used to create logs, enabling a way to trace back to transactions in the past, and information regarding who confirmed a certain action. There are ways of incorporating authentication by using a two-factor system such as a USB-stick or a Smart Card, which is what we will be exploring throughout this thesis. The physical media of choice for our implementation will be the YubiKey, which allows wireless connectivity through NFC as well as USB-connection.

As previously mentioned, the demand of secure computer systems with more sophisticated authentication methods with two or more factors is incessantly increasing. By developing and exploring this area we strive to find and pinpoint the areas which needs to be considered and can be expected to be encountered when creating a similar system. We will also discuss potential pitfalls and areas that might be restricted and therefore can limit the possibilities for certain systems.

On top of this we will also measure how this may affect the performance of the system. This will be evaluated by looking at the potential network overhead created by adding
additional functionality for logging in and digitally signing transactions with the use of the YubiKey.

Research Questions
To make a fair assessment and judge the general necessity we need to take a step back and look at each part of the implementation from a broader perspective.

- RQ1: What are the potential difficulties and drawbacks when implementing a two-factor authentication and digital signature system with a USB-device?

When answering the above question, we need to assess the route we chose and put it in context with other optional methods. What pros and cons did we consider with regards to security, flexibility and availability.

- RQ2: What kind of network overhead is created from verifying data and validating certificates?

The network overhead will be measured by fetching the amount of data (bytes) and packets that is sent between the client and server during a transaction. To compare the result with a baseline, we will run the system without the Digital Signing, as well as with it enabled.

Limitations
Our main objective is to generalize each step of the implementation of a system using a USB device for authentication and signing documents. However, we need to consider we are limited to only using the YubiKey, and other media might work slightly different. YubiKey is as close as it comes to be a “branch standard” security key, but we still need to make reservations for both co-existing devices as well as future alternatives.

BACKGROUND
The thesis work will be conducted at IFS¹. The exploration of alternatives is used for evaluation of a product that could be implemented to the IFS business platform in the future. A possible solution will benefit both employees and managers since the digital signatures can be used to determine who has signed a certain transaction and possibly detect if a document has been altered after it has been signed. The use of OTP would also increase the general level security of the system by adding another factor when signing in.

THEORY
Two-factor authentication (2FA) is an additional layer of security to a service on top of the classic Single-factor method (username and password). Instead of directly accessing a service after entering the password, another prompt appears where the user needs to provide a second piece of information often categorized as²:

- Something you know (PIN code, secret question)
- Something you have (Smartphone, YubiKey)
- Something you are (Fingerprint, iris scan)

We will be looking at the second option in the form of Universal second factor (U2F) where a second-factor device acts as a security key [3]. This method, where a single unit with multiple certificates also increases the convenience for the user, whom only has to carry a single device instead of different depending on the service or system they intend to use. In the private sector the method of using several layers to increase security was developed in order to stop attacks which targeted the communication between two parties [3]. These attacks are often referred to as man-in-the-middle attacks, and implies methods for fetching password by phishing and/or key logging software.

Increased security from outside attacks is of great concern for enterprises as well, but additional security checks and authentication can also be used to create checkpoints where the employee must identify themselves again. This increases the security within the work space and minimizes the risk of someone accessing another employee’s computer if left unlocked in an open office space.

Certificate Authority
To have full control over all certificates that has been issued a Certificate Authority (CA) is needed. This system relies on the Public Key Infrastructure (PKI) and on a chain of trust between all CA’s [4]. The system itself is only as secure as its weakest CA. The CA ensures that a certificate is valid before signing and makes sure to revoke a certificate that should no longer be in use. In a smaller closed system where the certificates only will be used inside a closed network a local CA will be enough to uphold this control.

For a bigger scope where a service must be available from any browser in any network the company must rely on buying the CA service from a trusted party, to issue certificates and control revoking them [5]. In the case of our study and developing the mock-up system we chose not to focus much effort into this area. We issue the certificates used locally for ourselves, since they are only being used in a very small and controlled environment and will not have any impact for the results of the paper. This will of course have to be taken into consideration if you were to implement a similar solution in a real system. Most bigger enterprises will already have certificates issued within the local environment, and if not, a reliable third party could supply a solution to issue and control certificates.

OTP
One-time passwords are a generally known term in the field of computer security. The OTP is generated by touching the button located on the surface of the YubiKey (Figure 1). The OTP consists of a 44-letter string. The first 12 are unique for that key and serve as the identification number while the rest are randomly generated each time you press the button. The full generated OTP are sent to Yubico’s verification cloud service for validation. The validation algorithm, and how to

¹ https://www.ifsworld.com/us/
² https://authy.com/what-is-2fa/
implement it is described in Yubico’s developer site [6]. This cloud service is not obligated to use if, for example, the YubiKey will be used in an internal system or offline system. You can always host your own validation service.

**Digital Signing**

To digitally sign and verify signatures, a key pair that is connected to the certificate on a certain user’s YubiKey needs to be generated. The key pair consist of a private key and a public key. When generated, the private key is stored on the YubiKey and is not accessible from the outside. The public key can be stored locally on a file or in a database and can safely be shared. The process of signing a document or transaction with this method then are:

- Encrypt the data with a hash
- Apply the private key to the hash to create a signature

Anyone with access to the public key can verify that a document is signed by a specific person by:

- Hashing the original data with the same algorithm
- Use the public key to decode the signature
- Compare the hash created in step one with the hash from the decoded signature

The full flow of how this could work in a system is shown in Figure 2.

Since the key pair only responds to each other, no public key other than the one generated together with the private key can be used to verify the signature. Every modification made to the original data would result in a different object when hashed. This implies that if either the data has been compromised after the user has signed it or the public key does not correspond with the signature, the verification will fail. On that ground, if the verification does not fail, we can safely assess that the person related to the public key in question indeed has signed off on the data, and no alteration has been made ever since.

**Yubico PIV Tool**

The Yubico PIV Tool is used to access and modify the Privilege and Identification Card (PIV) of the YubiKey. The PIV Tool can easily be used through the Command Line Interface (CLI), which allows small bash/bat scripts to be used when issuing and handling certificates. The standard for handling and creating cryptographic tokens is called PKCS#11[7]. This interface is used by the PIV Tool to access the USB device to enroll new certificates or modify existing ones.

**Network overhead**

Measuring network traffic between different parts of a system has been a vital part of benchmarking and measuring efficiency as long as systems has been operated over networks. In earlier days controlling and measuring data sent within a network was vital due to the severe limitations of the network capacity. This is of course relevant today as well, with a more environmental thinking that is spreading and is encouraged within the field of computer science [7]. When measuring network overhead there are several parts that might be of interest to look further into. One may be interested in the round-trip time (RTT), and in this case the number of turns taken between the server and client is of great importance. Another aspect, which is the one we chose to focus on, is the amount of data sent between the client machine and server. By looking into this we will find out how much data (in bytes) the extra layer of encryption and digital signatures adds to the transactions within the system. The increased amount of data may also result in more packets sent between the client and server when performing a certain task.

**RELATED WORK**

There are several options when it comes to choosing a method for two-factor authentication. Lang et al. [3] identifies strengths and weaknesses through comparing different U2F options in relation to each other. One option is to use a One-time passcode (OTP), which can be delivered by several different manners such as SMS or a separate wireless device [3]. This method increases the security, but the threat of man-in-the-middle attacks still remains as the generated code may still be used by the attacker, even if the passcode has a short expiration time. Using a smartphone as a second factor also comes with limitations. The phone has to be secured from malware, as well as the developers has to constantly update software to fit different platforms. Another possible method is using a smart card solution. Smart cards exclude the threat of phishing as the physical device itself needs to be present in order to access your system, and is not dependent on batteries. A severe limitation comes with the need to have proper equipment installed at the workstations such as a card reader [3]. By using Universal Serial Bus (USB) all of the above mentioned problems can be avoided. The interface is commonly used all over the world, and is preinstalled on most computers. The portability of USB-devices is also a great factor for the usability, it can conveniently be attached to a keyring for example.

**U2F Design**

When registering a U2F device (or security key) to a service on the web, the server of the service sends a challenge through the browser client according to challenge-response authentication protocol. The client then makes it a hashed data structure that together with the origin of the site is sent to the security key that is plugged into the USB–port [3].

The security key then generates a private key that the service will receive a corresponding public key of according to Public key cryptography. It also sends a key handle in the response to the service. A key handle is used to connect to a generated key instead of directly linking it to the service.

---

3 [https://www.oasis-open.org/standards/pkcs11-hist-v2.40](https://www.oasis-open.org/standards/pkcs11-hist-v2.40)
This due to the fact that the same security key should have the possibility to be connected to multiple user accounts on the same service. When the security key has sent over its response to the challenge from the service in form of a signature consisting of the earlier mentioned public key, key handle, origin and hashed data, the service will verify the signature and connect the public key and key handle with the logged in account [3].

**YubiKey**
The YubiKey is currently a popular type of security key and has had a lot of attention in the development of U2F [8]. It is the core product of Yubico⁴, a company founded 2007 in Sweden that are involved in creating solutions for internet security and is designed based on the structure that were described under U2F design [3].

A strength in this product is its development platform and the functionality seem to offer more options in comparison with its competition. Documentation for developers is available and suggestions for implementation on third party platforms is supported to some extent. The primary reason that speaks for YubiKey in terms of functionality is the possibility to apply digital signing of data by physical media. Tools to generate key pairs and certificates to support these operations are provided by the manufacturer as well as cloud services for validation.

![YubiKey 5 NFC](image)

**Figure 1. YubiKey 5 NFC**

**METHOD**
The goal for this thesis is two parted. The first part is to explore the possibilities, evaluate and propose which methods and tools are suitable for this project to find a proper solution. The other is to measure potential overhead that is to be expected from the extended security checkpoints within a U2F-system.

To start off the research phase we will investigate the options provided by Yubico themselves. This means exploring both the options and methods described from the official user guides and suggestions for implementations as well as looking at the developer platform⁵. After exhausting the options provided in this first step we will broaden the scope of research to other sources. This includes both looking at relevant published literature and more informal information such as forums and blogs where similar topics are being discussed. This to gather as much information as possible and gain a broader perspective before moving on and have some basis before making decisions for the implementation.

**Exploration**
We need to thoroughly investigate potential software that can be used to fully utilize the YubiKey. By doing this we can draw conclusion on what can be used for our project from other sources and what needs to be developed partly from scratch. We don’t expect to find fully functional libraries, but rather find solutions that can be moderately modified to meet our needs.

Apart from Yubico PIV Tool other options were explored to manage certificates, such as the YubiKey Manager, which uses a graphical interface to issue and handle certificates as well as other security key functionalities. This option did not meet our needs, due to the fact that we were severely limited when creating and modifying certificates and key pairs. However, the manager offers functionalities which may be useful for the end-user, such as changing PIN/PUK-codes and viewing certificates.

**Digital Signature**
The process of signing a transaction or document consists of adding the private key onto the object in question. This requires suitable tools to be found in terms of extracting the private key from the YubiKey and then applying it to the transaction to generate a signature. Extract in this case meaning “use”, since a private key cannot be extracted to say a text file for example because it is generated to be secret. To grasp the practical approach to do this, the first step to understanding was to install OpenSC, a CLI based tool that allowed you do these types of operations and be in complete control over the content of the YubiKey in a comfortable fashion. Even though this was not an appropriate tool for the final web-based system, it was possible to both sign files and verify signatures using this tool, which led to that the practical process of digital signing became clearer. This were however only a tool for understanding. For the mockup system, something that would work in a web browser was needed. As the mockup system was built in JavaScript to have a web-based interface, the issue when it came to digital signing was support of client-side cryptography in browsers [8].

To enable the possibility to execute a signing operation in a web browser we looked at what options there were to implement such a way. One often referred solution was a

---

⁴ [https://www.yubico.com/](https://www.yubico.com/)

⁵ [https://developers.yubico.com/](https://developers.yubico.com/)
third-party application called Fortify\(^6\). Fortify is a locally installed third-party application that runs on the client machine. When initiated, it listens to a TCP port and communicates with Fortify’s WebCrypto polyfill\(^7\) that is written into the JavaScript. Fortify makes it possible for the web applications to use the functionality of smart cards (or as in our case, a security key).

**Implementation**

The first step of implementation will be to create a smaller and more restricted environment. We will setup a node server with a simple login form consisting of username and password. A user database in json will also be created to house the users with access to the system. This environment will be relatively simple, and its purpose is to evaluate methods where the two main aspects that will be investigated is:

- Logging in (entering the system with the help of the physical media and complimentary certificates).
- Signing a transaction, in connection to the user that has already identified themselves when logging in.

We do not intend to implement this on an actual live system, however, the certificates used represent real user information connected to a database in the testing environment.

**Logging in with OTP**

OTP is an appropriate method to use for the login operation. The user database is therefore going to be expanded with the YubiKey identification numbers so each key can be associated with a specific user as a second verification method. A field in the login form will also be added for the key input. When a user logs in, in addition to the password control, you’ll press the YubiKey to generate an OTP.

**X.509 certificates**

X.509 certificate is the standard format for PKI and handling digital certificates on the web. The certificates are used over safe internet protocols such as https. When creating a new certificate, the certificate is signed by the CA to be trusted and issued to a specific user. To create and manage these certificates Yubico PIV Tool will be used. A small bat/bash-script will be used to create the key pair and certificate for each new user.

\(^6\) [https://fortifyapp.com/](https://fortifyapp.com/)

\(^7\) Polyfill is an expression for code that enable web browsers to use features that they normally do not support.
Digital Signing

To setup an experimental case in the system where the user that logged in would sign a mock transaction, a landing page for the user was created to arrive at after login. In this view, a text input field appears where the entered string is to represent the transaction that are to be signed. When the sign-button is pressed, a digitally signed object will be created, that can be transported to the server.

When using Fortify, a socket is opened and a connection to a chosen TCP port is established. Inside this connection calls can be made to predefined functions from the WebCrypto that makes it possible to do cryptographic operations such as encrypt, decrypt, get keys and certificates, perform signing and verifying operations.

After a signature is completed a proper verification function needs to be implemented. The purpose of this is to enable a server side verification for the transaction before storing it to the database. Secondly, by enabling this all previous transaction that has been made can also be verified at any given time. In practice this means we need to be able to list transactions, and then by selecting a specific transaction we can fetch the public key related to the transaction and use it to verify that nothing has been altered after the signature was applied.

Measuring Network overhead

The overhead will be calculated in relation to the operation of making a transaction. In our case this means we will send a text-string from a client to be saved to the database on the server side. This measurement will be done with the implementation we have created, meaning the text will be encrypted and signed before being sent to the server. We will then run an identical setup with two exceptions:

- The encryption and digital signature will not be applied to the transaction
- The server will not verify the transaction once it arrives (since there is nothing to verify)

This will enable us to determine how much overhead the digital signature and verification creates, by comparing the results of both methods. The size of the transaction (number of characters in the string) may have an impact on the result. This may be circumvented by having a static number of characters as input in the text field when running the tests. We have decided to measure this by doing tests with three different sizes of the text input: 5, 150 and 2000 characters. Each sample size will be run 5 times in each setup to create an average for each instance. This will assist us in drawing further conclusions with regards to whether the overhead is growing exponentially, constant or even has a diminishing effect with growing text size.

We will measure the size of all packets that is sent between the client and the server in conjunction with a transaction.

The number of turns communication is transmitted between the client and server will also be measured and taken into consideration when doing an evaluation of the result. All of this will be measured by following the TCP stream of Application Data with the help of Wireshark8.

RESULTS

This section is divided in a similar fashion to the structure of the method. Results of the two key functionalities used under implementation as well as the measuring of the data when creating a transaction both with and without digital signing will be presented.

As we began to explore different services and tools that could be used to handle the YubiKey, we realized that there is different functionality implemented on the YubiKey suitable for different operations. For example, for login operations, One-time passwords appeared to be the best solution whereas for digital signing, certificates and keypairs had to be generated and used in order to apply it in a correct and secure way. In the case of digital signing, tools such as Gnu Privacy Guard (GPG) and OpenSC were examined but discarded due to the fact that they are CLI based. They also seemed to be more tedious to implement compared to the solution we ended up using if to be applied onto a system that were to be web-based. GPG did however supply web plugins for web-browsers but not precisely what we were searching for.

OTP

To make the YubiKey function as user authentication with OTP, everything does exist from the start as long as you are using Yubico’s cloud service for validation. If you are using this service, you have a dependency both on having an internet connection and that the cloud service is fully functional and up and running. The OTP solution provided by the YubiKey is very easy to use and implement, if you have a database of users which can be modified without causing other parts of the system to malfunction. To achieve a fully functional OTP validation within a closed network further implementation needs to be done. This would however give the system owner a greater control of the system itself and increase its flexibility.

Certificate Creation

Firstly, we generate a key pair to the corresponding slot, as in our case will be slot 9c which will hold certificates associated with digital signing. Options regarding policies for PIN-codes can be set during this step, which was not possible through the YubiKey Manager, as well as preferences for the encryption. Next, we created a self-signed certificate to which we then applied the public key generated in the first step. The certificate is stored locally before being exported to the YubiKey. During these steps a default PIN will be set, which will be used for digital signing. This can

---

8 https://www.wireshark.org/
be reset and changed by the user whenever they want with the help of YubiKey Manager. See figure 3 for an example of a certificate viewed in the YubiKey Manager.

![Certificate](image)

**Figure 3. Certificate viewed in YubiKey Manager**

During the second step the public key will also be exported and saved to the user database. The public key is then always associated with the correct security key holder. This will be used when verifying a signature by the server.

**Digital Signing**

The biggest problem in implementing a cryptographic digital signature system is to make it work in a web environment because of the restrictions of today’s browsers. The main concern comes with regards to enabling the browser to utilize the certificate stored on the USB-device. This access must be done to use the private key which is stored on the device and cannot be exported from it. For security reasons this can not be accessed by anyone without permissions. Therefore, the easiest way to circumvent this is to use a third-party application that runs on the client machine that has permission to access and use the private key stored on the device.

In our case we chose to use Fortify. By calling the appropriate functions from the WebCrypto that is opened by Fortify, the certificate linked to the current user is automatically selected and the correspondent private key is extracted and applied to the transaction to create the final signature (figure 4).

```javascript
const key = await GetCertificateKey("private", provider, {$("certificates").value});
if (!key) {
    throw new Error("certificate have no private key");
}

const alg = {
    name: key.algorithm.name,
    hash: "SHA-256",
};

const message = $("message").value;
const signature = await provider.subtle.sign(alg, key, message);
```

**Figure 4. Code block depicting signature creation**

**Verification**

When the signing is completed, the signature along with the original data is base64 encoded to strings and sent to the server. There it gets decoded and verified by using NodeJS built-in library called “crypto” that provides cryptographic functionality for the NodeJS server. Through this library, the signature can be verified so that it can be determined that the signature when transported from client to server did not get compromised.

The verification process works by looking through the user database to match the user that has performed the signing of an object and extracts the public key belonging to the user. The public key, original data and signature is then sent to the verification function that the crypto library provides (Figure 2).

Finally, if the verification is approved server-side, the base64 encoded signature and data is stored in a signature database under the user that performed the signing (Figure 2). The same function can also be used in a later instance when a signature must be verified. In that case however, instead of using the current signature that just has been signed, the signature that is to be verified is selected and retrieved from the signature database in order to apply the lookup in the user database.

**Network overhead**

The results of measuring the network communication between a client and the server when making a transaction shows that the overhead is close to being static in size. This means that the smaller the size of the transaction, the greater the relative overhead size will be (Figure 5). The number of turnarounds between the client and the server is always doubled for the digitally signed transaction since the server has to validate and return a success or unsuccessful response.

---

9 base64 is a common way of encoding binary data to string format
in conjunction. This will lead to an increased time consumption for each transaction, especially if the server is remote and has a high RTT.

Figure 5. Table showing overhead generated from adding digital signature to a transaction

DISCUSSION
In the following section we will discuss the results and methodology used in the thesis. Each part will be discussed separately starting off with the result section, which is divided into separate sections for each topic which was introduced in the previous section.

OTP
There is no need to install third-party applications or extensive tools to generate OTP or use the YubiKey for this purpose. This however means that everything is on default settings and YubiKey Manager or PIV Tool must be installed to configure these settings. Even though OTP is much safer than regular single factor authentication there is still many flaws in its level of security [10].

Digital Signing
The functionality of allowing cryptographic digital signing with the YubiKey in a web-browser without the help of a third-party application is not an easy task. As the technology of the browsers is today, they limit the use of cryptography in a web environment. Even though solutions exist in circumventing this, it requires a lot of knowledge, time and resources to do. Because of that, there is a market where companies supply these products for purchase. If, however you want to do it yourself, you will most likely have to depend on a third-party application that runs on the client machine that does the signing.

If you’re not dependent on signing in a web-browser and are satisfied with signing files in a CLI, there are simpler ways where probably the most straightforward and easiest approach is using OpenSC.

Either way you will have to use Yubico PIV Tool to setup keypairs and certificates to enable the use of YubiKey for the signing and the YubiKey Manager to configure the pin code and managing certificates.

Network overhead
The result is to be expected by the way we have implemented our transactions. The signature is created by hashing the message, using SHA-256 and then applying the private key. This will result in a signature of the same size regardless of the message (transaction) size. All data will be base64 encoded before being sent from the client to the server, in both cases (with and without signature). This means that the only difference between the two transaction types is the addition of a base64 encoded signature in the case with digital signing. The additional data and round trip between the server and client when validating the signature will also generate a transaction of static size.

The sample size we collected for comparing the two transaction models is relatively small. This due to the problems that occurred when trying to generate a greater size, which was our initial goal. When making several transactions within a short time frame some of the data were cached and thus generating results that was not accurate. This resulted in us having to manually make transactions and then reset or wait long enough for connection between the client and server to be totally lost. Gathering the data from Wireshark was also somewhat tedious and time consuming. Apart from cached data resulting in corrupt data, other instances such as lost packets and sudden disappearance of SSL certificates (Probably cached in some way as well) made for a lot of frustration in gathering useful data.

Method
Since one major part of the thesis is exploratory, the methodology in some cases may be hard to replicate. The method itself with regards to this will therefore probably yield different solutions and findings in the case of a future study similar to this. This not only due to the fact that newer and hopefully better tools will be available but may also be due to a different approach to the implementation itself.

Results
The path chosen for the approach that eventually became the final one in developing the prototype, was decided partly because of the lack of progress concerning this particular field. The technology in digital signing with an USB security key has not been out for more than for a few years as of
that was the easiest way was to use OpenSC that we had experimented something that did generate th
an experimental solution because we wanted to get and the line arguments, taking in the variables from the user session

The first web-stage, the first thought wa
Because the access to all NPM command line arguments in the NodeJS environment.

There was also the potential issue of implementing a solution for this case to make a functioning prototype.

There was also the potential issue of implementing a solution that was browser based. Since web-browsers are limited in this sense and does not allow client-side JavaScript access to certificates, keys and smart cards, we also had to find a way around that restriction.

Because we used OpenSC earlier when exploring the possibilities regarding usage of the YubiKey, we began there. There were Open SC command line arguments that could be used to sign, store and verify files. This worked with the help of pkcs11 and pkcs15. As this still was in an early stage, the first thought was to implement execution of command line arguments in the NodeJS environment. Because the access to all NPM packages existed thanks to our test environment, the thought seemed possible.

The first web-based solution was to execute the command line arguments, taking in the variables from the user session and the YubiKey to create the signature. This was more of an experimental solution because we wanted to get something that did generate the result we were after and the easiest way was to use OpenSC that we had experimented with earlier.

```javascript
function signTransaction(id) {
    let transObj = 'transactions/tid_' + id + '.txt';
    let pin = 123456;
    // HASH TRANSACTION //
    exec('openssl dgst -sha256 -binary ' + transObj + ' > tmp/curTransHash.sha256');
    // SIGN TRANSACTION //
    exec('pkcs15-crypt -s -t tmp/curTransHash.sha256 -p ' + pin + ' -o signedTransactions/signed_' + id + ' .output -f openssl --sha-256 --pkcs1 -k 02');
}
```

Figure 6. OpenSC signing function (unsafe and simplified)

Beside the unsafe use of command line arguments, another issue was that every signature or item that were to be signed, was saved to files. Because of that we got stuck with two folders, one with original documents that had been signed and one with the signatures of those documents. Both needed to be saved due to verification. This made for a lot of files to keep track off after a while when the folders started to grow. We realized that this was not viable in the long run. Not only for the sake of keeping tabs on all the files that were generated. But also, because the signatures were to be stored in a database, not on file as well as the transactions that were to be signed, necessarily weren’t files either. OpenSC did not seem to support this as far as we could see. So, we had to move on from this idea and started to look for libraries that supported the functionality we were after.

It certainly would have been more optimal to find a solution not depending on the need to install an application on the client machine to make it work. As mentioned earlier, this is recent development. On that note and because it is both difficult and expensive to develop, which probably is the reason why there is a market to sell these services, it seemed like the best path to stay on. This meant having to find a third-party application. In our case that became Fortify.

This application did not have a lot of documentation or information about it. The only things that really existed for our benefit were its website with quite concise information of the product and a repository on git without documentation, just a few examples on how to implement. Luckily, one of the examples that was provided was close to what we wanted to achieve, so that was a breakthrough in getting it to work in a browser.

There is always a question regarding how encrypted data has to be when it comes to security. We did not encrypt everything such as the public key or the YubiKey id or even the user password which of course always should be encrypted when stored. This only for the reason of simplicity

10 https://www.yubico.com/about/yubico-innovation-history
11 https://www.npmjs.com/
while the work was progressing so in this situation, that would not be a problem but if it were to be used on a real system this of course would be mandatory.

**CONCLUSIONS**

In this thesis we explore the options for implementing a second factor, in our case the YubiKey, to be used for authentication and digital signatures. 2FA is growing in popularity when it comes to security solutions for both private use and business enterprises. A Common solution is using a smartphone as the U2F device. This project was a way of investigating the possibilities of using a USB-device as the second factor to sign transactions and look at how this platform works from a developer’s perspective.

Even though we restricted ourselves to the prototype outside of the business platform in question, the results may be useful from an enterprise point of view. The results suggest that there are a few chokepoints that a developer must overcome to create a successful implementation to create digital signatures with a USB-device. The main one being able to effectively access the certificate on the USB-device without relying on a third-party application that enables this utility. This restriction mainly comes from the improved security provided by modern web browsers, which limits the use of cryptographic operations.

**Future Work**

Because we limited ourselves to the prototype, the obvious step forward would be to implement the solution on an existing system to make it possible to sign proper transactions. Preferably the system at IFS that we had in mind when starting the project.

Another direction that can be taken in moving this project forward is to try to keep the same functionality not depending on another application running on the client machine.

Other factors that is not connected directly with this thesis work would be to look at more primitive problems that occurs with making a system depend on having a second device. Things to consider may be forgetting the key at home, losing it in general and creating new certificates if the current one would get blocked. These could be measured in lost time for the employee as well as the employer who would lose revenue/income from having their employee unable to work.

**REFERENCES**


