Two factor encryption in cloud storage providers using Hardware Tokens

Paul Crocker
Instituto de Telecomunicações
Universidade da Beira Interior, Portugal
Email: crocker@di.ubi.pt

Pedro Querido
Instituto de Telecomunicações
Universidade da Beira Interior, Portugal
Email: a21802@ubi.pt

***Abstract*—Clouds are widely used for storing, backing up and sharing information. Data confidentiality and privacy issues are important and topical issues in the evolving cloud technology. In this paper we describe a system that addresses the issues of securely accessing and storing data in cloud repositories. This paper will describe a two factor encryption architecture for cloud storage that incorporates the use of a hardware token. We have validated experimentally this architecture by developing a middleware that can be used with any cloud storage provider that makes use of the OAuth 2.0 protocol for authentication and authorization. The middleware enables the use of two-factor authentication and encryption mechanisms to ensure the privacy of the data, in this paper the YubiKey USB cryptographic token is used as the external two-factor module.**

I. Introduction

Concerns about security, privacy and compliance issues are important barriers for enterprises adopting cloud technologies. Also, with the increase number of user identities accessing cloud resources over a vast range of devices, the difficulties in managing passwords and keys also increases. Although Cloud Storage Providers (CSP's) may offer service contract guarantees that all the stored information is not accessed either by the CSP administrators or by malicious users or intruders, there is often no real mechanism to actually prevent them doing so.

A possible approach to resolve this problem would be creating a secure container where the user's files are added and which only the user controls. Using a master key derived from a user chosen passphrase, the user could encrypt and decrypt this secure container and access the files, however there are some drawbacks with this implementation. For example, if an attacker is able to guess the user's passphrase, all the files stored in the container are compromised. Another drawback is that even if only a single file is changed, the whole container must be synchronized with the one existing in the Cloud environment.

Another approach to this issue is a system that encrypts each file individually instead of using a secure container. Although the synchronization issue is then resolved, the first drawback would still be present since an attacker would be able to gain access to all the files once the user's passphrase is compromised. This approach does however have proponents, such as the BoxCrypt [2] application.

In order to protect against these kinds of attacks, a second factor of encryption of a different nature could be used. This way even if the user's passphrase, which is something memorized, is obtained by an attacker for instance via a keylogger, social engineering or any other mechanism, the attacker would still lack the access to the second factor required to derive the encryption key, usually a physical device. This has the added benefit of simplifying or standardizing authentication and key management. The rest of this article explores this idea and presents a proof of concept based on this concept using the YubiKey hardware token and the CSP product of the the largest telecommunications service provider in Portugal known as MEO Cloud.

The popular YubiKey USB token is usually used only as an authentication token and not as part of any encryption process (except as a static password container). This paper describes how to leverage such tokens in a general sense and extend their use and application to create a two factor encryption process. We retain the authentication stage, where a One Time Password is generated by the token, which is used together with the user's credentials in order to obtain access to our system. Files are then encrypted using a custom key, derived from a user passphrase and a key stored in the YubiKey token. Using the Near Field Communication (NFC) capabilities of the YubiKey, this system can also be used by any mobile user with a smartphone equipped with this feature. The middleware can also be accessed using a REST API for which the user must also be authenticated with his/her YubiKey token.

In this paper we propose a two-factor key derivation scheme using an hardware token and present a proof of concept that enables this approach in a cloud storage scenario, assuring end-to-end encryption and, therefore, the privacy of the users' data. The paper is organized as follows. Section 2 describes background and related work. Section 3 details the Two-Factor Encryption Scheme. Section 4 describes YubiCrypt, a proof of concept of our Two-Factor Encryption Scheme in a cloud storage environment. Section 5 presents conclusions and future work.

978-1-4673-9526-7/15/$31.00 ©2015 IEEE

II. Background and Related Work

A. Yubikey

YubiKey [3] is a small hardware device developed by Yubico and when connected to a USB port it emulates a standard keyboard It is compatible with most operating systems and uses the USB connection for power supply, thus no battery is required. It is capable of performing a series of cryptographic operations, such as generating a One Time Password (OTP) or performing Challenge-Response operations. It can also be configured with a static password. YubiKeys have two configuration slots [10], allowing simultaneously two different configurations, all the end user needs to do is plug it into a USB port and press the touch button on top of the device.
Regarding OTP generation, the YubiKey is capable of generating standard OAUTH (Open Authorization Protocol tokens as well as *Yubikey-OTP*, a standard developed by Yubico. For Challenge-Response operations, are made using either the HMAC-SHA1 [4] standard or the YubiKey OTP standard [8].
There are several different YubiKey models available. The YubiKey Standard and YubiKey Nano offer the same feature set described above where the key difference is that YubiKey Nano has a considerably smaller size. The YubiKey NEO model, as well as having the same features as the other models, is also capable of being accessed using Chip/Smart Card Interface Device *CCID* and is also able to communicate using the NFC contact-less technology.

1) Use Cases and Existing Applications: A lot of solutions have integrated YubiKey as a two-factor authentication provider, such as for computer logon, single sign-on and password managers.
One example worth mentioning is with *LastPass*, which is a cloud-based password management solution. It allows users to store all their credentials securely so that they can have complex passwords without the issue of having to remember them. All this information is encrypted using a master password and synchronized across all the browsers that a user may interact with. *LastPass* has integrated YubiKey as another factor of authentication. With this feature enabled, in order to access the users personal password database of passwords, the user must authenticate with the usual username/password combination and must also provide a YubiKey generated OTP. Another use case of the YubiKey is the open-source solution *Password Safe*, another password manager. Unlike LastPass, Password Safe is a client application and the password database is stored locally on the user's computer disk. Password Safe uses the YubiKey Challenge-Response feature to derive a key used to encrypt the database file.
Other solutions exist such as for disk encryption, for instance via the static password configuration or Challenge Response Mode for Full Disk Encryption Products. Finally a recent application,developed by Yubico, allows for example Windows users to use their YubiKey as a Time-based One-time Password (TOTP) token.

B. Client Side Encryption

An example of a popular application that enable users to encrypt files on their cloud service storage providers at the application client is Boxcryptor. Boxcryptor is an application that enables users to encrypt their data before uploading it to the cloud. It creates a virtual drive where users can place their files. These files are then encrypted and placed on the cloud storage application folder. The cloud storage client application handles de file synchronization task.
Boxcryptor uses AES with a key length of 256 bits, CBC (Cipher Block Chaining) and PKCS#7 padding to encrypt the files. The AES encryption key is derived from a password using the PBKDF2 algorithm with HMACSHA512, 10.000 iterations and a 24 byte salt. Each user has an RSA key pair which they and not BoxCrptor manage, every file encryption key is then encrypted with the clients RSA public key and then stored along with the encrypted file. In order to decrypt a file, the (encrypted) AES encryption key is retrieved along with the file. This key is then decrypted using the RSA private key and then used to decrypt the original file [2].
There are several other popular applications that use client side encryption or end-to-end in the same manner, for instance SpiderOak (www.spideroak.com) and myCryptoVault (www.mycryptovault.com). However none of these solutions incorporates the idea of a two factor hardware token as part of the process.

III. Two Factor Encryption Scheme

A. Encryption Key Derivation

In order to be able to encrypt files using two factors, a Key Derivation algorithm was designed. This key stretching algorithm was developed with the two-factor principles in mind in which the user provides two means of identification - one is typically a physical token, in this case the YubiKey USB cryptographic token, from now on simply referred to as YubiKey token, and the other is typically something memorized, in this case a passphrase. To derive an encryption key, a random 265-bit sequence is first generated. This sequence is then sent to the YubiKey token and then, using the private key configured in the device, the YubiKey computes an HMAC-SHA1 with a length of 160 bits. A key derivation algorithm, such as PBKDF2 with SHA-1/SHA-2, is then used in order to derive the final encryption key. This key derivation algorithm is also used to derive a key to be used in the File Integrity Verification process, described later in section III-B1.
In this process, shown in Figure 1, a user supplied passphrase is used as the input and the YubiKey computed HMAC-SHA1 is used as the salt. The minimum number of iterations used in our implementation, using PBKDF with SHA-256, is 64000. However, this number is not fixed

and can be later increased in order to accommodate the increasing processing power of computers.
The initial approach investigated involved modifying the PBKDF2 [7] algorithm so that the HMAC-SHA1 [6] computation was performed externally by the second authentication factor, in this case the YubiKey token. However testing shows that even for a small number of iterations this computation would take a long time, due to the overhead introduced by the USB communication and by the lack of processing power of these types of tokens. In fact we found that for 1024 iterations, a Key Derivation operation would take more that 60 seconds. For the same number of iterations, the new approach described above is able to perform the same operation in about 100 milliseconds, while preserving the desired security level.In addition, our novel approach allows our Key Derivation Algorithm to have a higher degree of modularity. This means that the underlying Key Derivation Function can be chosen from a set of supported algorithms, such as PBKDF2 and scrypt [9]. This modular approach also enables our solution to be used with other tokens such as the Plug-up AuthentiKator device [1].

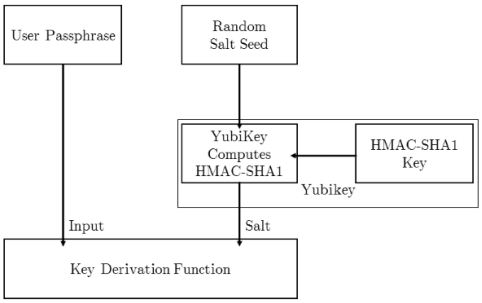

Fig. 1. Encryption key derivation steps.

B. YubiCrypt Files Specification

Defining a file format is a very important step as it establishes a clear ruleset for the layout of the contents of a file so that the reading and writing processes can be implemented. If the chosen format is made public, other application developers are able to open and create files that are compatible with this implementation. Given the fact that additional data needs to be stored along with the encrypted bytes of the original file, defining a concise way to structure this information was imperative and so the YubiCrypt File Specification was created. This specification - Version 1 - was designed with extensibility in mind and is believed to be secure, in the sense that, without knowledge of both the passphrase and secret key residing in the user's YubiKey, only a brute-force attack or flaw in the underlying cryptographic algorithm or implementation will result in unauthorized access to the encrypted data. There are some public settings that must be saved during the encryption process so that they can be reused in the decryption process. In order for this information to persist, and to add flexibility to this solution a header is added to the beginning of every encrypted file. A description of the header format used in this application can be found in Table I.

Version	Cipher	Salt	IV	It	Validation
4 bytes	4 bytes	32 bytes	16 bytes	4 bytes	32 bytes

TABLE I
YubiCrypt File Specification header.

The `Specification version` field contains the version of the YubiCrypt File Specification being used. It is the sequence of 4 bytes, where the first 3 are *0x59, 0x43, 0x46* (the characters "YCF") followed by 1 byte indicating the version being used, in this case version *0x01* of the specification. This is a quick way for applications to identify a file as a YubiCrypt File.

The `Cipher Suite` field indicates the cryptographic algorithms used to derive a key, encrypt the file, and verify the integrity and authenticity of the file. It is a 4 byte sequence, where the first byte indicates the Key Derivation Function used to derive the encryption key, the second byte identifies the symmetric cipher used to encrypt the file, and the third byte identifies the MAC algorithm used to verify the integrity of the file. The fourth byte is reserved for future use.

The supported Key Derivation Functions for version 1 include PBKDF and scrypt for key derivation, AES (CBC and GCM Mode), Serpent TwoFish, ChaCha for the symmetric cipher and various Mac Algorithms.

The `Salt Seed` field is a sequence of 256 random bits generated before the key derivation process. These bytes are used as the HMAC-SHA1 input computed by the Yubikey token. The 160-bit output is then used as a salt for the encryption key derivation described in section III-A.

The `IV` field is a 128-bit random Initial Value used by the Block Cipher in order to encrypt the file.

The `It(erations)` field represents the number of iterations to be performed by the encryption key derivation function described in section III-A. This value is stored in order to *future-proof* the file format against increases in processing power. Increasing the number of iterations to be made when deriving the encryption key also increases the need for CPU power in order for this function to complete in a reasonable time. This way, brute-forcing also becomes harder do achieve, given the properties of the underlying Key Derivation Functions [7].

The `Validation` field is a 265-bit SHA-2 digest of the passphrase selected by the user concatenated with the USB Tokens unique serial number. This field is used to verify that both factors used to derive an encryption key are supplied during the decryption process. Hence, if an incorrect passphrase is entered or the incorrect USB token is plugged in, the decryption process will not even begin, thus saving time and resources.

1) File Integrity: If a simple symmetric cipher is chosen to encrypt the data, a message authentication code (MAC)

is appended to the end of every encrypted file. This keyed-hash MAC (HMAC) is calculated over all the bytes of the encrypted file, including the header.Integrity and authenticity can also be guaranteed when an authenticated encryption algorithm is used. In this case, the authentication tag is appended to the end of the file.

C. File Encryption Process

Each file is encrypted using one of the algorithms available in the currently supported cipher suites. All the block ciphers currently use a 256-bit key and operate in CBC mode with PKCS #7 padding.

IV. Proof of Concept: YubiCrypt

In this section we describe YubiCrypt, a proof of concept middleware constructed to demonstrate the process described in the previous section. Currently the YubiKey USB cryptographic token is used as the external two-factor module. At the authentication stage, an OTP is generated by this token and used together with the user's credentials in order to obtain access the system. The system is integrated with the ASP.NET Identity Framework, which recently enabled the use of two-factor authentication factors such as email or sms tokens. Files are encrypted with a key that is derived from a user pass-phrase and a key stored either in the YubiKey token or on a Key Storage Server. Using the NFC capabilities of the YubiKey, this system can also be used by any mobile user with a smartphone that has this feature. The middleware can also be accessed using a REST API for which the user must also be authenticated with his/her YubiKey token. Client applications that make use of the YubiCrypt middleware have been built for Windows and Windows Phone.

A. YubiCrypt Online

YubiCrypt Online is an ASP.NET MVC application with a RESTful webservice developed using the Web API framework. It is responsible for the management of users as well as their YubiKey tokens. It is also possible to store the YubiKey's secret key in the YubiCrypt platform so that the Encryption Key Derivation process can be performed without using the physical device, enabling users to access their files using the YubiCrypt Windows Phone application. YubiCrypt Online is the necessary middleware between the user's devices and a cloud storage provider. YubiCrypt Cloud is divided into four main components: the Web Client, the Validation Server, the Key Storage Server and the RESTful webservice.

1) YubiCrypt Web Interface: The Web Interface component is the browser based client in which a YubiCrypt user can manage his/her YubiCrypt account. This component allows users to manage their YubiKey tokens - configuration files, generated during the provisioning of a YubiKey can be uploaded to the system enabling users the use their newly configured YubiKey with the YubiCrypt platform.

Users must grant authorization for the YubiCrypt platform to partially access their cloud storage account in order to use the platform, an access token and access secret are then passed to the YubiCrypt platform so that they can be used in the future to access the user's Cloud without requiring him/her to login to that storage provider. The user can of course, at any time, revoke this access.

2) YubiCrypt Validation Server: The Validation Server component is responsible for authenticating and authorizing users using their YubiKey token as a second factor of authentication, and is required in order for a user to access the YubiCrypt web interface and the YubiCrypt RESTful API. It was implemented according to the Yubico OTP specification [8]. This is our own implementation of a validation server and was developed so that it can be tightly integrated with the whole YubiCrypt ecosystem.

3) YubiCrypt Key Storage Server: The Key Storage Server component exists mainly to enable Windows smartphone users to encrypt and decrypt files. Windows Phone devices are unable to communicate with YubiKey devices to perform HMAC-SHA1 Challenge-Response operations. Given this limitation the key derivation process cannot be completed, hence to overcome this issue, users can opt-in for storing an encrypted copy of their secret key in the Key Storage Server. In this scenario, when accessing the files stored on the YubiCrypt platform using a mobile client, the Key Derivation process would still be possible, given the fact that the HMAC-SHA1 that would be calculated inside the YubiKey token is instead calculated in the device itself.

Note that storing the HMAC-SHA1 Challenge-Response secret key in the server does not give anyone with access to the system the possibility to decrypt any of the files. The Encryption Key Derivation process, as described in III-A, is specifically designed to require both this secret key and a user selected passphrase. As an additional security measure, this key is also encrypted before being transmitted from the client to the server. When retrieved, the user must have the decryption key in order to use it.

Storing this key online is also useful for safety reasons. If a user loses his/her YubiKey, it will be impossible to decrypt the files. By having this key available elsewhere, this data can still be recovered given that the user is able to provide the second encryption token, namely the passphrase. In the event that a user forgets the passphrase used to encrypt a file it will not be possible to decrypt the file. For enterprises with extra security requirements the online key can be split or shared amongst many servers using well known key splitting or secret sharing algorithms.

4) YubiCrypt RESTful API: This component is the endpoint that is used by all the client applications in order to interact with the YubiCrypt platform. All the requests to the YubiCrypt REST API must be authenticated. For this purpose we use the OAuth 2 protocol [5] so that the API can decide if the request is authorized. In order to make a request to the API, a client must present an access token.

Client applications must first request an access token before they are allowed to access the API. Every issued access token is valid for 20 minutes, a time span that we believe to be able to accommodate a normal usage session. Every time this access token expires, client applications force the user to re-enter his/her credentials so that a new access token can be issued.

B. YubiCrypt Client Applications

Currently there are two YubiCrypt Client Applications available, for the Windows Desktop *YubiCrypt Desktop* and the Windows Phone *YubiCrypt Mobile*. All Client Applications use the REST endpoint exposed by the YubiCrypt Online to execute file operations.

1) YubiCrypt Desktop: YubiCrypt Desktop was developed using C# and WPF. The main functionality of the application is to enable the user to access all their (encrypted) files stored in the YubiCrypt platform, as shown in figure 2. Files can be uploaded and encrypted to the cloud or downloaded and decrypted from the cloud. The application can also be used in *offline* mode permitting users to encrypt and decrypt local files. Users are also able to view all their current active authorization tokens with their cloud storage service providers, this is also shown in figure 2. The application also allows users to provision their YubiKey.

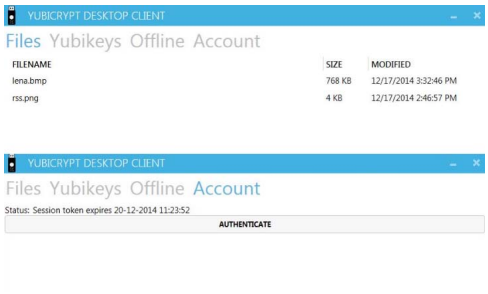

Fig. 2. YubiCrypt Desktop listing the contents of a folder and Current Authorization Details

2) YubiCrypt Mobile: YubiCrypt Mobile is a Windows Phone 8 application that enables users to access their encrypted files stored in a cloud environment from any NFC enabled Windows Phone device. The application uses the NFC capabilities of the Yubikey NEO to perform a two-factor authentication with the YubiCrypt middleware.

C. YubiCrypt Online Authentication

In order to login into the YubiCrypt Online platform, a user must first input a valid username and password, then the user must provide a second factor of authentication, a Yubico OTP generated by his/her YubiKey. If this token is successfully validated then the user is granted access.

It is also possible to use providers of two-factor authentication other than the YubiKey device. Having this option is mostly important in two scenarios:

- When the user first needs to login and has yet to register a YubiKey to be used with the YubiCrypt platform;
- If the user loses his/her YubiKey but still needs to access the YubiCrypt platform.

With this in mind, two other providers of two-factor authentication were added: Email and SMS token. After validating his/her credentials, a user can choose to receive an email or message containing a TOTP that must be used to complete the authentication process. Thanks to a recent update to the ASP.NET Identity framework, enabling these providers required little effort given the fact that they are provided by the framework, all that needs to be handled is the delivery of the generated TOTP.

Due to the extreme modularity of the ASP.NET stack, it was possible to integrate the YubiKey two-factor authentication directly into the ASP.NET Authentication pipeline. It is believed that this is one of the first two-factor authentication extensions to the ASP.NET Identity framework and quite possibly the only one to do it so using the YubiKey.

D. YubiCrypt Sequence Diagrams

The following sequence diagrams allow a better and more concise view of our solution described previously. Figure 3 shows a sequence diagram of the steps required to connect a YubiCrypt account to a Cloud Storage Provider and how to obtain an access token in order to make API calls. The first step is the typical cloud client third party application phase of requesting authorization from the CSP, this is normally made only once and is valid whilst the user does not revoke access. Figure 3 also details authentication to our application using the Yubikey.

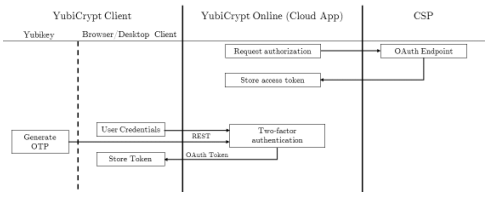

Fig. 3. YubiCrypt Authentication Flow Diagram.

Figure 4 details the flow when encrypting and uploading a YubiCrypt protected file to a Cloud Storage Provider. In particular it shows the key derivation process using the Yubikey Token and the interactions with the various OAuth tokens when uploading the file from a client application to the YubiCrypt Online Application and then to the respective Cloud Storage Provider.

E. Server-side data protection

Some sensitive information is stored in the YubiCrypt database, specifically the MEO Cloud access token and, in some cases, the encrypted HMAC-SHA1 secret key. If an attacker is able to gain access to the database, the retrieval of this information could facilitate an attack to the system.

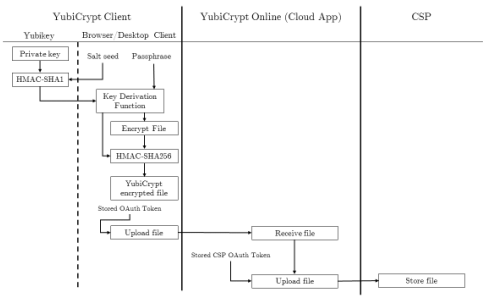

Fig. 4. YubiCrypt File Encryption and Upload Flow Diagram.

Retrieving the MEO Cloud OAuth access token and secret would allow the attacker to have full control of the user's YubiCrypt encrypted files stored in this provider. Note that the attacker would not be able to access files and folders that the user may have other than the ones inside the YubiCrypt designated folder neither would it be possible for the attacker to decrypt the files. However an attacker would still be possible to delete all of the user's encrypted files. This flaw is mitigated given the fact that most cloud storage providers keep a backup of the data. Restoring one of these backups would revert the effects of this type of attack.

On the other hand, retrieving the encrypted HMAC-SHA1 secret key, if present, would not allow an attack. First the attacker would have to be able to decrypt this key. In a scenario of a successful decryption of this key, due to the nature of the Encryption Key Derivation scheme, it still would not be possible to decrypt any of the files using only the retrieved HMAC-SHA1 secret key because the passphrase is also needed.

V. Conclusions and Future Work

The YubiCrypt platform presented in this article leverages all the features offered by secure USB security tokens, such as the YubiKey, and enables them to be used in a Cloud Storage scenario in order to create an extra layer of security and privacy. The Yubicrypt platform presented is a broad implementation that verifies the two-factor encryption concept using commonly held two-factor authentication tokens in order to enable end-to-end encryption in a cloud storage scenario.

Future development of the YubiCrypt platform includes the integration of more Cloud Storage Providers as well as making the API available for others to implement their own client software. In terms of mobile usability we have developed a hybrid solution for the windows mobile platform and in the future will develop an Android Application (that uses NFC). For the iOS platform we have no actual plans due to the current lack of NFC support. Finally a virtual drive could be implemented on users' computers enabling seamless upload and download of encrypted files. Although the name YubiCrypt is a clear reference to the YubiKey token, we are also working to extend support for other USB cryptographic tokens.

A. Availability

The full source for our proof of concept is publicly available under the MIT License. The Git repository is located at https://github.com/PedroQ/YubiCrypt.

Acknowledgment

The authors would like to thank the support of the RELEASE laboratory, UBI, the Instituto de Telecomunicações, the Soft SIM Project Portugal Telecom and the Fundação para a Ciencia e Tecnologia (FCT) Portugal UID/EEA/50008/2013.

References

[1] Authentikator technical details. http://authentikator.com/index.php/fiche-technique/. (Last accessed 26th November 2014).
[2] Boxcryptor technical overview. https://www.boxcryptor.com/en/technical-overview. (Last accessed 26th November 2014).
[3] Yubikey hardware page. https://www.yubico.com/products/yubikey-hardware/. (Last accessed 26th November 2014).
[4] Frank Hoornaert David Naccache Ohad Ranen David M'Raihi, Mihir Bellare. HOTP: An HMAC-Based One-Time Password Algorithm. RFC 4226, RFC Editor, December 2005.
[5] Dick Hardt. The OAuth 2.0 Authorization Framework. RFC 2616, RFC Editor, October 2012.
[6] Ran Canetti Hugo Krawczyk, Mihir Bellare. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, RFC Editor, February 1997.
[7] B. Kaliski. PKCS #5: Password-Based Cryptography Specification. RFC 2898, RFC Editor, September 2000.
[8] Robert Künnemann and Graham Steel. YubiSecure? Formal Security Analysis Results for the Yubikey and YubiHSM. In Audun Jøsang, Pierangela Samarati, and Marinella Petrocchi, editors, *Security and Trust Management*, volume 7783 of *Lecture Notes in Computer Science*, pages 257–272. Springer Berlin Heidelberg, 2013.
[9] Colin Percival. Stronger key derivation via sequential memory-hard functions. https://www.tarsnap.com/scrypt/scrypt.pdf, May 2012. (Last accessed 26th November 2014).
[10] Yubico Inc. *The YubiKey Manual V3.1*, April 2013. Available at http://www.yubico.com/ (Last accessed 26th November 2014).