

YubiKey Security Evaluation

Discussion of security properties and best practices

Version: 2.0

Date: 9th September 2009

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design, and manufacturing. Yubico shall have no liability for any error or damages of any kind resulting from the use of this document.

The Yubico Software referenced in this document is licensed to you under the terms and conditions accompanying the software or as otherwise agreed between you or the company that you are representing.

Trademarks

Yubico and YubiKey are trademarks of Yubico AB.

Contact Information

Yubico AB
Kungsgatan 62
111 22 Stockholm
Sweden
info@yubico.com

Contents

1	Document Information	4
1.1	Executive Summary	4
1.2	Audience	4
1.3	Background	4
1.4	Related documentation	4
1.5	Document History	4
2	Introduction	5
3	Device and Protocol Description	6
4	Definitions	9
4.1	Server attack	9
4.2	Protocol Attacks	9
4.3	Host Attacks	9
4.4	Device attacks	9
4.5	User Attacks	10
4.6	Other Attacks	10
5	Protocol Security	11
5.1	Cryptographic strength	11
5.2	Server strength	12
6	User attacks	13
7	Device and Host Security	15
7.1	Hardware key extraction and side-channel attacks	15
7.2	Software key/token extraction	15
8	Likely Attack Vectors	16
9	Conclusions	17
10	References	18

1 Document Information

1.1 Executive Summary

We describe Yubico's YubiKey user authentication device and the protocol it uses. After discussing general attack vectors, we consider some possible attacks on the system. The security of the system relies to some extent on the design of the server, and we discuss these requirements. The system uses strong encryption with AES. Replay attacks are handled. Trojan and phishing problems are mitigated. Combined with a PIN or password the system provides two-factor authentication. We believe the system balances good security with ease of use, and the threats to the system are few and well-understood.

1.2 Audience

This document is intended primarily for readers with a technical/research IT background. The document assumes knowledge of basic security concepts and terminology.

1.3 Background

The initial version of this document was written in 2007 by Simon Josefsson working as a third party security reviewer. A few months later Simon joined the Yubico team and since then the document has been updated to reflect the latest development of the YubiKey.

For this reason the document is in large parts still written in the form of an "independent" evaluation of the YubiKey. We believe this approach allows us to identify and discuss each aspect from a balanced viewpoint.

1.4 Related documentation

The YubiKey Manual – Usage, configuration and introduction of basic concepts

1.5 Document History

Date	Version	Author	Activity
January 2007	1.0	SJ	Initial version
September 2009	2.0	SJ	Cleanup and updated for YubiKey 2.0

2 Introduction

The YubiKey is a small device carried by a user and its purpose is to authenticate the owner against some service. While the device may be used for various authentication purposes, our focus will be on authentication to remote Internet websites.

In this paper we describe the device and the protocol it uses to communicate with the server, we analyze the security of the system, and we discuss overall system security concerns.

3 Device and Protocol Description

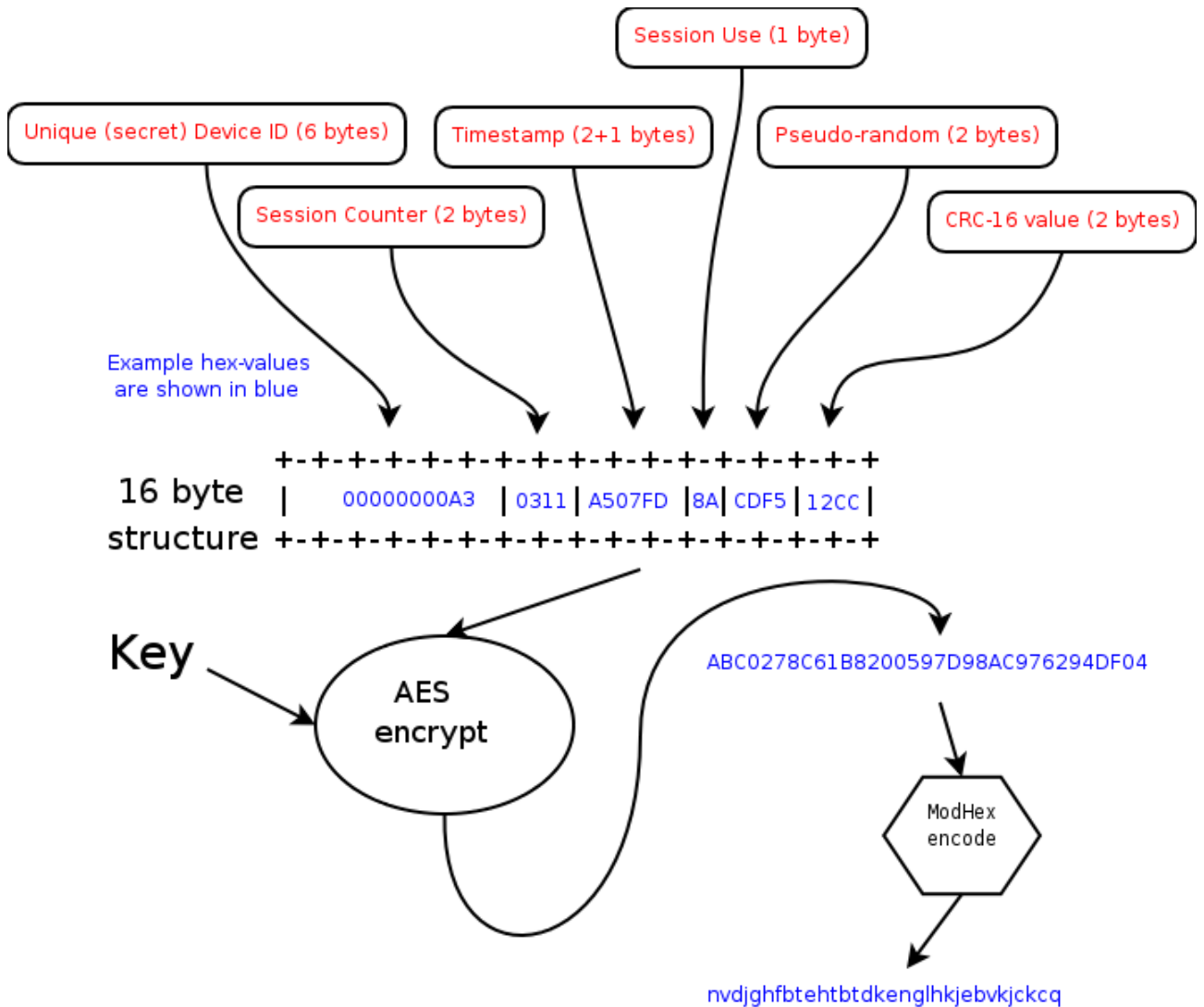
The user connects the device to his or her computer through the USB port. The device receives power through the USB connection, and does not need any external power supply or a built-in battery. By emulating a USB keyboard, no device drivers or other special software needs to be installed in the computer. The device is equipped with a touch button that triggers an authentication attempt. When the button is touched, the device generates a plaintext authentication token, encrypts the token using its unique per-device key with the [AES] algorithm, and then encodes the ciphertext with "ModHex", a new variant of base16 [BASE-ENC] encoding.

The YubiKey 2.0 introduces a mechanism where the user can use two separate credentials. We call the storage for these credentials "slot 1" and "slot 2". To generate a credential from slot 1, the user touches the button for a short period of time (e.g., well below 2 seconds). To generate a credential from slot 2, the user touches the button for a long period of time (e.g., well above 3 seconds). With proper user education, we believe this does not add any additional security problems so we continue to evaluate the YubiKey configured with just the slot 1 credential.

The device is independent of the operating system, hardware and application, as long as the operating system supports USB Human Interface Device (HID). Popular operating systems such as Windows 2000/XP/Vista/7, Mac OS X, GNU/Linux and Solaris support USB HID.

This control flow is shown in the illustration below. The resulting textual token is transmitted to the computer by emulating a USB keyboard and simulating the keyboard inputs of the characters in the encoded token.

The intention is to use the token with a web service. The user sets the keyboard input cursor on an HTML input field and touch the button. The token is sent by the browser to the remote server using HTTPS. The token is used to authenticate the user for that session. Services will often also request a username and password from the user, but this will vary depending on the threat model.



The ModHex encoding is used instead of standard hex or base64 encoding to make the device independent of language settings in the operating system. Some keyboard layouts switch some keys, for example, German keyboards swap Z and Y. The alphabet “cbdefghijklnrtuv” is used because these characters were found to be unswitched on all keyboards.

The plaintext token is 16 bytes long (the same size as one AES block) and consists of the following fields, in C form.

```

#define UID_SIZE 6
typedef struct {
    uint8_t userId[UID_SIZE]; /* Unique (secret) ID */
    uint16_t sessionCtr; /* Session counter
    (incremented by 1 at startup */

    uint24_t timestamp; /* Timestamp incremented
    by approx 8Hz */

    uint8_t sessionUse; /* Times used within session */
    uint16_t rnd; /* Pseudo-random value */
    uint16_t crc; /* CRC16 value of all fields */
} TICKET;

```

The 6 byte userId is unique to each device and is hard coded when the device is personalized for a particular user. The field size allows for about 100000 billion unique devices.

The 2 byte sessionCtr is stored in non-volatile memory (i.e., it is preserved when the user unplugs the USB device). It is initially set to 1. It is incremented every time the device is used after power-up. (The counter does not wrap; when it has reached 0xFFFF the counter sticks there. This corresponds to about 25 tokens every day for 7 years or 5 tokens every day for 35 years.)

The 3 byte timestamp is stored in volatile memory, and is set to a random value every time the device is connected. It is incremented by a 8Hz clock, which may drift up to 20% depending on temperature and characteristics of the device silicon and the power source. The YubiKey 2.0 has a lower drift ca 2%. The timestamp field tracks the length of time the device has been connected during a particular session. The resolution corresponds to about 24 days. The counter wrap around when incremented beyond its range.

The 1 byte sessionUse field counts the number of authentication tokens generated during the particular session. If it wraps, the sessionCtr is incremented by one.

The 2 byte rnd field contains "random" values generated by a LFSR register seeded by the touch button sensor USB activity. (Note that although the source is reasonable random, it cannot be assumed to provide cryptographically strong random numbers.)

The 2 byte crc field contains the 1-complement of the ISO 13239 CRC-16 checksum computed over the entire token excluding the CRC field itself. When verified over the entire token, the checksum shall match the fixed value 0xF0B8. (Note that it is not used to cryptographically authenticate the plaintext, it is only used as a check to detect data corruption.)

4 Definitions

To structure our discussion of attacks against the device, we will first define a few classes (and sub-classes) of attacks. The classes correspond roughly to different modus operandi of the attacker. For example, if a device is stolen, we talk about a *device attack*, and if a server is physically compromised, we talk about a *server attack*. Advanced readers may skip this section.

4.1 Server attack

A **server attack** is where the attacker interacts with the server to gain unauthorized access. The attack can be a **physical server attack** if the server machine itself is compromised, e.g., the machine is stolen, or the attacker gains unauthorized physical access to the machine. The attack is an **indirect server attack** if the attacker remotely accesses the server over the Internet. We further classify the indirect attack into the **cracker attack** where the attacker is permitted to use any method to gain access to the machine, in particular software vulnerabilities in unrelated system software. The other interesting indirect attack is a **research attack**, where the attacker analyzes the protocol used, and finds a weakness in it. For example, the attacker may connect to the server, pretend to be a valid user and through this gain access. The research attack is also covered by the protocol attack, see below.

4.2 Protocol Attacks

A **protocol attack** is where the attacker interacts with the server to gain unauthorized access. A **research attack** is described above. If the attacker passively listens to traffic between the server and a legitimate user, we call this **traffic analysis**. If the attacker listens and is able to modify traffic, we call this a **man-in-the-middle attack**.

4.3 Host Attacks

A **host attack** is where the attacker targets the computer that the user uses. Threats include **key loggers**, which record all keyboard input. Other attacks include redirecting network traffic. A **buffer overflow** attack against the USB stack in the device is also possible.

4.4 Device attacks

A **device attack** is where the attacker gains access to a device for a particular user, and uses this access to gain unauthorized access to the service. The **unnoticed device attack** is where the attacker temporarily gains access to a device and the authorized user never detects the attack, and will possibly continue to use the device. The **destructive device attack** is where the attacker steals the device to pick it apart to gain some advantage. A third category is when the authorized user actually is the attacker, and attempts to use his device to gain unauthorized access as another user or to the system itself, which we will call the **fraudulent user device attack**.

4.5 User Attacks

A **user attack** is where the attacker targets the authorized user, to trick him into revealing information that helps the attacker gain unauthorized access. A **phishing user attack** is where the attacker sets up a web site that gives the impression of being the real server, in order to collect authentication information. A related scenario is the **social engineering user attack** where the user is tricked into giving the attacker the authentication information, e.g., over the phone.

4.6 Other Attacks

It is clear that an attacker may combine two or more of these types of attack, and if he or she gains more than what can be expected from the individual attacks, we call it a **combined attack**. An attack where two or more legitimate users combine the capabilities of their own devices to gain access to other users or the system is called a **colluding attack**.

5 Protocol Security

5.1 Cryptographic strength

The YubiKey uses the 128-bit block length AES cipher with 128-bit keys in ECB mode to encrypt authentication tokens. The key is assumed to be unique to each device since that is recommended best practice.

The plaintext is easy to recognize with high probability, thus, a naive brute force attack to exhaust the entire key space would require in the order of 2^{128} operations. This is well beyond the range of today's publicly available technology.

The ECB mode is vulnerable to codebook attacks, which is avoided because the same plaintext is never encrypted twice. This is because the plaintext contains a unique static per-device field "userID" and the 2 byte session counter that is incremented for every powerup and the field "sessionUse" that is incremented for every authentication token. Thus the combination of sessionCtr and sessionUse will never be the same for two different tokens.

The use of 128-bit blocks in AES and the unique plaintext make it possible, regardless of the cipher mode employed, to distinguish cipher text from random values after (on average) 2^{64} blocks are encrypted with a single key. This is well above the maximum number of times a device can be used (limited by the session counter/use fields giving a max of 2^{24}), and thus not applicable.

There are fairly generic pre-computation attacks against all block cipher modes that allow a meet-in-the-middle attack. These attacks work by generating and searching huge tables of cipher text associated with known plaintext and known keys. If the memory and processor resources are available for a pre-computation attack, the strength is limited to around 2^{64} operations. The use of a non-predictable component in the plaintext significantly increases the size of the table that the attacker must compute to mount a successful pre-computation attack. The relatively small number of encrypted tokens that will be available to an attacker further decrease the probability of a successful attack. We believe the work load involved is well beyond the adversaries we are considering. See [SCHNEIER, p.358].

The protocol relies on AES in ECB mode for message integrity (to guarantee that the user id and session counter fields are not tampered with) and message authentication (to guarantee that the sender of the token is the person in possession of the encryption key). AES was designed to provide privacy, but it offers no provable message integrity or authentication. See section 1 and 2 of [ROGAWAY] for a discussion of these topics. We note that other token based authentication systems, such as the RSA SecurID (with the AES algorithm), also use AES in ECB mode. Given that the same plaintext is rarely encrypted twice, we conjecture that, if an attacker can discover the key, or be able to provide a valid encrypted token for a particular user, given only a (limited) number of earlier encrypted tokens, this indicates that the AES block cipher has unwanted properties.

Replay attacks are not possible because the server will store the last successful sessionCtr/sessionUse and compare it against new incoming tokens. If the token has a lower sessionCtr/sessionUse than what is stored on the server, the OTP is treated as a replay.

5.2 Server strength

The server stores the encryption keys for all users, and is thus a single point of failure. A successful attack on the server, e.g., a **physical server attack** or **cracker attack**, will compromise the AES keys for the YubiKeys. The attacker will be able to masquerade as any user if one-factor authentication is used, but needs to also recover user's password if two-factor authentication is used. The recovery procedure from a server attack includes replacing all user devices (or possibly to do a software upgrade of them).

The encryption keys may be stored in a tamper-proof hardware box, such as a IBM 4758 or a more recent similar device, connected to the server, and the external device perform decryption of the incoming tokens. The attacker will then be unable to extract the encryption keys. This improves the situation when a server is temporarily compromised by a cracker attack; not all user devices will need to be replaced. However, it does not protect against a physical server attack.

6 User attacks

The behavior of the user in a security system is always a risk factor. There are several ways to behave as a user that puts a system at risk. This area is difficult to discuss in a technical or scientific way. To fully understand the importance of all issues, one would have to study a (large) group of people and log: when, and how, they use the system, and draw conclusions from that study. Even then, one can never be certain that all the risks have been fully evaluated. For many end-users, the user is the weakest link in the system.

The classic **social engineering** approach of talking (often over the phone) to users and asking them (after tricking them into believing they are talking to the operator of the real server) for their username and password is not a big concern in this system – the length of the authentication token (32 seemingly random looking characters) leads us to believe that few will have the patience to read the string (not even considering reading it correct when reading such a long string). We note that having the system to ask for new tokens throughout a session will detect if this has occurred (the timestamps will be off).

A face-to-face social engineering attack, where the attacker borrows the device and acquires the username and password will succeed, and there is no protection against this (although see below on biometric device authentication).

Another scenario is the **phishing attack**, where the attacker manages to get the user to send the tokens over the Internet to the attacker. The attacker would also have to acquire the username and password too, which reasonably means that the attacker somehow has tricked the user into believing he or she is talking to the real server. (We assume that users will not send the username, password and authentication tokens to unrelated sites, although this may be a simplification.) The attacker can achieve this by utilizing well-known weaknesses in Internet protocols such as DNS or BGP. Using HTTPS may reduce this risk if users are aware that they should not use the server if it does not utilize HTTPS, but studies show that few users understand this. (Using HTTPS helps to protect usernames and passwords, which improves privacy and prevents passive traffic analysis.)

Once the attacker has acquired the username, password and authentication tokens, he or she may replay them to the real server. This will only work during the time window from the attack to when the user uses the token to the real server the next time, because the server will detect an inconsistent session id.

By asking for multiple tokens during a session, the server can force the attacker to do a real-time attack, and thus defend against attacks where the server collects username, password and authentication tokens over a longer period of time and then uses them at once. This is the anti-phishing idea in the YubiKey. Real-time phishing is when the user is tricked into handing over his credentials to the attacker, and the attacker uses the credentials in real-time against the authentic system, and this is not protected against.

A real-time phishing attack can be attempted on a system wide level, and for a multi-million user organization, simple statistics says that even if only 1% of the users fall for the phishing scam and are tricked to go to the server, and if only 1% of them actually log in to the server and send several tokens, the attack will succeed against hundreds of users.

We conclude by noting that the system does not provide good defense against a real-time man-in-the-middle or phishing attack. It is likely that by having the system force attackers to do real-time attacks, some attacks are prevented, and it increases the chance of finding the perpetrator, through legal means, by chasing the IP addresses of the attacker.

(A solution for real-time phishing is mutual authentication. However, many existing systems that provide mutual authentication (such as smart cards) are vulnerable to other attacks (e.g., Trojans) or require that additional software is installed on every host the user uses. It is not clear that all schemes offering mutual authentication leads to better overall security than YubiKey.)

7 Device and Host Security

The security of the device itself is an important factor. The YubiKey contains a unique encryption key, and if an attacker can extract the key, he can compromise that particular user's account. Keep in mind that the attacker will also need to acquire the username and password.

7.1 Hardware key extraction and side-channel attacks

Picking the device apart and using instruments to directly or indirectly extract the encryption key stored in the device silicon would lead, together with the username and password, to a compromise of the user's account. This technology is available to well-funded attackers, typically government-level agencies. This step requires access to the device, knowledge of the username and password, and expertise and equipment. This approach will most likely also damage the device, which would be noticed by the real owner. After considering that this effort by the attacker only compromises one particular user, we believe this minor risk costs too much to protect against.

A side-channel attack is where the attacker uses information, such as the time taken to compute a token, and use that information to derive (for example) the encryption key. Because each token is triggered by a hardware button (its imperfection make it difficult to performing exact timings), and the relatively small amounts of encrypted tokens which are available, we believe this is not a feasible approach.

7.2 Software key/token extraction

If an attacker could control the device from software on the host, to either generate tokens or extract the encryption key, they could also mount an attack. This attack is complicated and somewhat speculative. However, because it potentially would allow attackers to mount an attack on many targets with small per-target costs, it warrants some consideration. The attack requires two steps.

The attacker must first take control of the host system that the user uses, for example by installing a Trojan, to be able to directly talk to the device. This step is not easy to carry out, but is within the range of the attackers we are considering. (Note that a username/password based system would have been compromised at this point.)

Second, the Trojan planted in the system must be able to extract the key from the device, or have the device generate tokens on command. Neither action is permitted by the design of the tokens. However, both hardware back-doors (e.g., for debugging purposes) and software flaws provide a possibility. If there is a hardware back-door or a flaw in the USB stack in the device that forces the CPU to trigger execution of a specific program address, or read data stored in the device, or similar, the attack seem feasible.

8 Likely Attack Vectors

As we have seen, the design of the device and a reasonable server prevents many known attacks. There are three possible attack vectors that we wish to identify and discuss further. We believe that any successful attack against the system would most likely use one of these three methods.

We used two metrics to identify these methods. The first is the economic and practical range of the attackers we are most concerned with. The second is the estimated gain of a successful attack. The attackers we are concerned with are moderately funded organizations with access to the publicly available state of the art knowledge in this area.

The first bar for any attack is thus to be within the attacker's range, which we believe excludes crypto-analytical attacks against AES, direct attacks on the server, or the special equipment and expertise required to read out the encryption key from a device. The second bar excludes attacks which only apply to a small number of users. This rule out, for example, an attack that gains access to the password and then physically steals the device.

The potential attacks that pass both bars are server attacks (see section 4.1 and 5.2), real-time phishing (see section 6) and remote encryption key or token extraction (see section 7.2).

9 Conclusions

The YubiKey system protects against many of the problems in username/password based authentication systems. For example, it protects against attacks that use key loggers or fraudulent software that collect passwords from computers or the network. The encryption algorithm employed is state of the art Advanced Encryption Standard (AES).

We have identified three likely main potential attack vectors: server attacks, remote encryption or token extraction, or real-time phishing. The first one is a well known problem that can be addressed with good processes and sound engineering principles. The second threat is speculative, and requires a conspiring manufacturer or the exploitation of a hardware flaw. The final threat, real-time phishing is our primary concern.

We note that there has historically always been a trade-off between ease of use and cost on one side and security on the other. There is to our knowledge no technology that solves the real-time phishing problem that is easy to use for the user and cost effective for an organization.

10 References

- [AES] Advanced Encryption Standard, FIPS PUB 197, <http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [BASE-ENC] The base16, base32, and base64 data encodings, 2003, RFC 3548, <http://www.ietf.org/rfc/rfc3548.txt>
- [SCHNEIER] Applied Cryptography, Bruce Schneier, 1996, ISBN 0-471-11709-9
- [AES-CCM] Counter with CBC-MAC (CCM), 2003, RFC 3610, <http://www.ietf.org/rfc/rfc3610.txt>
- [ROGAWAY] Problems with Proposed IP Cryptography, 1995, P. Rogaway, <http://www.cs.ucdavis.edu/~rogaway/papers/draft-rogaway-ipsec-comments-00.txt>