Secure Login without Passwords

T.M.C. Ruiter*

December 22, 2013

Any person can invent a security system so clever that he or she can't imagine a way of breaking it.[1]

Abstract

These are the ingredients for secure logins:

- The website maintains a small array with Secret keys in an HSM, which will be renewed regularly (oldest key removed, all keys shifted one position, new key added).
- All cryptographic operations at the website take place in the HSM; no values are remembered, except Secret keys.
- For each user, a 128 bit random number acts as the site key for that user. The user gets a different 128 bit value, which is the $\texttt{AES}_{256}$ encryption of the site key for that user with the newest Secret key.
- Logging in is a process of proving that both the website and the user have the right key by sending hashes of random 128 bit numbers encrypted with these keys. Neither the site key nor the user key are ever sent over the line.
- When Secret keys are changed, the user automatically gets a new key. This way, user keys are changed regularly as well.
- All user keys are put on a keyring, which is a set of at least 99 keys, each 128 bit long, most of them dummies. No other information whatsoever is stored in a keyring, which may be stored in an ordinary file. The keyring may be encrypted with its own keys.
- The user selects which key is used for what, which it needs to write down or remember. Although keys themselves are changed regularly, the key number and its purpose will never change during the lifetime of the keyring, which might be forever. This makes remembering key numbers, at least for those keys that are used regularly, doable for most people.
- The keyring itself is something you must have; the key number something you must know, so logging in using a keyring is a basic form of two-factor authentication.

*Martijn Donders has helped review the cryptographic functions.

[1] Also known as Schneier's Law.

Contents

1 Introduction

Passwords should be kept in memory (human memory, that is) at all times. This article is about passwords for websites; the passwords many people use on a day to day basis. Passwords are an archaic type of security measure, compared to the scheme proposed here.

1.1 On human behavior

The rationale amongst security experts is that passwords should not be written down. Ever. And passwords should be unique for each and every website. Oh, and—I almost forgot—you have to change them as well. Each month or so will do nicely.

Yeah, right! I cannot remember all different passwords I am forced to use, although my memory is quite good. I *have* to write them down, otherwise, I am lost. (Fortunately, I have some support for this [2].) I don't trust software that will "remember" my passwords for me, because their "memory banks" might be on a malicious server on the other side of the ocean. Therefore, I resort to a little black booklet, with all my account information. I don't remember passwords any more, I remember where my booklet is. And I confess that I am compelled to reuse passwords, and to keep the ones I am not forced to change, so I can actually remember some of them. So I believe nothing has fundamentally changed in more than 13 years... [1]

I have given up on inventing a scheme for passwords that differ from each-other, can be changed individually at different times, and are easy to remember, as not to be forced to write them down. I believe no such scheme exists, because websites have different requirements regarding passwords. Some don't allow spaces, some complain about length, some fuss about dictionary lookup. The interval at which you are forced to change passwords is different for websites; some changes are compulsory, some voluntarily, some just to stop annoying pop-ups.

For the user, the bad thing with passwords is that you have to keep track of it all. Remembering difficult passwords is cumbersome for most, and impossible for some. Tracking things infalliably, and remembering different passwords for each and every site is not something people excel at.

1.2 The keyring system

Using 128 random bits as a key to gain access to a website is far more secure than letting people decide which password they would like to use to do so.

Imagine a keyring, not unlike your own keyring on which you have keys for your car, your house, shed, or locker. This keyring has a label and 99 keys on it, numbered 1 through 99. Instead of brass or steel, these keys are made of 128 random bits each. The label is another 128 bits long, and as random as possible, like the keys. Instead of being on a steel ring, these keys, with the keyring label, are written to a file on disk; a blob of 100 strings of 128 bits.

The use of keys on this keyring is not entirely different from using real physical keys. The bits in a key are comparable with the tooths or holes of a physical key you use to unlock your home. You do not need to remember exactly how far the tooths need to protrude or exactly where and how deep the holes in your key need to be, to be able to unlock the door; you just select the right key (the whole physical thing at once, with all the right tooths or holes) by recognising its form or its label. And

with a physical key you cannot open a door; you can only unlock it. You still have to push...

The use of this proposed keyring has several advantages over the current practise of websites to use passwords for logging in. Instead of having to remember dozens of passwords for numerous sites, you only need to remember a key number for that site, in the range of 1 to 99. This key number stays the same for that website at all times, so you *can* remember it.

1.3 Advantages

In the following cases a keyring is superior to the use of passwords.

- Websites have all login information stored centrally. If an hacker can obtain this data and decrypt it, it has access to all—possibly millions—accounts at once [3].

  Using the keyring system there is no usable login information whatsoever at the server hosting the website. There is no way that the user information that *is* present yield any usable login data. Hacking a website to obtain logins is useless.

  Other reasons to hack websites will remain, however, and using keyrings does not prevent hacking; it just eliminates one of the major attractions.

- Users tend to have the same password and the same login name for several websites. A hack of an insufficiently protected site could yield valid usernames and passwords of perfectly protected sites. (Hack of www.babydump.nl yields at least 500 valid logins for www.kpn.nl.)

  Even if all user keys for a website were obtained in a hack, these would be useless for any other website, since they differ by definition.

- Websites require the user to change passwords. As more websites do this, more and more passwords a user has to remember, change. Ideally, no password for a website should be the same as for another website, but that is impractical. This would mean that each and every password needs to be written down, because the number of passwords is too much to remember for most. This thwarts the principle that passwords need to be remembered and never written down. The requirements to change passwords frequently and that they should differ from any other password is an inhuman task.

  Using the keyring system, keys will differ for each website by definition and change regularly and automatically. Key numbers (the ordinal numbers in the keyring) don't change, so most of them can be remembered by the average human.

- Sometimes, getting unauthorized access to an account is as simple as just looking at the keyboard to see what the password is. The userid is always displayed when logging in, so shoulder surfing is very effective.

  Using a keyring, shoulder surfing cannot be used directly to login. Since a keyring is something you have to have, you cannot login using only the userid and the key number. You need to have access to the (unencrypted) keyring as well. Therefore, using a keyring is a basic form of 2-factor authentication.

- People tend to use weak passwords (unless a website specifically enforces the use of strong passwords) which can be guessed using specialized tools. If that yields no success, brute force attacks can be launched; to just try all possible passwords with limited length.

  Password guessing nor brute force attacks are an option when trying to login, since no passwords of any kind are exchanged. Even if the keys themselves would be used as an old-fashioned password, the search area would encompass 2^{128} or $3.4 \cdot 10^{38}$ equally likely possibilities. Trying 1 million possibilities per second it would still take 10^{32} seconds to try every possibility.

- The validity of the connection to websites is built on trust. HTTPS connections are protected using certificates. Sometimes trust only goes so far, and bogus but valid website certificates are used (Dorifel virus) or even the Root CA certificates are forged (see the DigiNotar hack). In that case, the user's trust is betrayed and the user left helpless.

  With the keyring solution no standalone substitute websites can exist; login data must be redirected to the real website. A substitute website does not have the right Secret keys. A user will notice this by wrong answers from the website and the login will abort from the user side.

- Visitors of websites are lured to other, well built fraudulent websites, mimicking websites of banks and such (phishing). Here, a simple mail can give a lot of trouble, redirecting users to an unsecure copy of a website, without the user suspecting anything.

  All communication to and from the malicious website can be passed on to the real website, to give the user the sense it is talking to the real site. Obtaining valid login data this way (as a man in the middle) is useless, since no keys are sent over the line. The data that is used to validate a user is meaningless for the next login.

The use of a keyring with random numbers provides a basic two-factor authentication. The user has to have possession and access to its keyring. Furthermore, knowledge is required about which key is used for what (and which key is used to encrypt the keyring itself) to be able to use the keyring.

2 Keys

The exchanged values in the login process and the keys in a keyring are 128 bits long in this article, but could be taken much smaller; say, 16 bits each, without doing great injustice to the level of security. The use of 128 bits is only a convenient size, as it is the $\texttt{AES}_{256}$ blocksize, and $\texttt{SHA}_{256}$ works with this length naturally. This way, we take full advantage of their cryptographic power.

2.1 Secret keys

The keys used to encrypt other keys—called Secret keys in this article—need to be secret, inaccessible and unobtainable. Therefore, they are kept in a Hardware Security Module (HSM). The Secret keys are used by the HSM to perform $\texttt{AES}_{256}$ encryptions of site keys, and are 256 bits long.

There are multiple Secret keys stored in the HSM. They are considered to be stored in array S, in this article. With this array, three values are defined.

1. The number of stored Secret keys is represented by m.
2. The constant[2] `MAX_KEYS`, which is the maximum number of keys that will be kept. The value of m starts at 0 and will reach `MAX_KEYS` eventually and grow no more. The value `MAX_KEYS` has a minimum of 2 (a new key and at least 1 old key). All values in the array S are shifted when a new key is stored, which will always be inserted at $S[0]$. This way, with at least one key loaded ($m > 0$), it always holds that $S[0]$ is the newest key, and $S[m-1]$ is the oldest key. With $m > 1$, $S[1]$ is the jongest old key.
3. The constant `MAX_ACTIVE_KEYS`, which is the maximum number of keys that will be used for logging users in. This value lies in the range $[2, \texttt{MAX\_KEYS}]$.

   The subtracting `MAX_ACTIVE_KEYS` from `MAX_KEYS` gives the number of inactive keys which can be used to reactivate expired user keys. Users using a user key that is older than the oldest active Secret key cannot login, but their key can be restored with an inactive Secret key. If a key is used that is older than the oldest inactive key, the key cannot be restored and is lost forever.

2.2 Site keys

For each user, there exists a site key and a derived user key (see section 2.3). The site keys are stored in a table of a database, where the $\texttt{SHA}_{256}$ hash of the userid will act as the primary key for that table. Since the user only sends the $\texttt{SHA}_{256}$ hash of its userid, no actual userids will be stored.

Site keys are generated once by the random generator of the HSM, and need never be replaced.

Each time a user tries to login, the database is queried with the $\texttt{SHA}_{256}$ hash of the userid, and the site key for that user is returned for further processing.

2.3 User keys

A user key is computed by encrypting the site key for the user with the joungest (or current) Secret key, using the $\texttt{AES}_{256}$ algorithm.

The keys users get from the website are stored on a keyring (see section 3). These keys are automatically changed for new keys by the website when the Secret key is changed.

2.4 Key lifetime and old keys

For enhanced security, Secret keys need to be replaced at regular intervals. This change of keys is initiated by the website, without the possibility to make individual arrangements with any of its users.

After changing the Secret key, it is no longer possible to login with any user key. Therefore, an array of Secret keys needs to be kept, allowing users to login using an

[2] Although declared a constant, the value of `MAX_KEYS` may change over time. When the login policy regarding the use of old keys is changed, more or fewer old keys will be stored. For this, `MAX_KEYS` needs to be changed.

old(er) Secret key. At least one old Secret key needs to be stored; otherwise, the Secret key cannot be changed without rendering all user keys permanently useless.

Upon successful login with an old key, the user will be provided with a new user key, with which it can login using the newest Secret key, from that moment on. Changing of user keys is seamless and the user might, just as well, be kept unaware of this.

The website's security policy should prescribe with what frequency Secret keys are to be changed, and how many old keys should be kept. If, for instance, the Secret key is changed every month, and 11 old keys are kept, users can still login using a key that is a year old. If the user key is older than that, the site is not able to verify the user's key any longer.

2.5 Compromised keys

There are several scenarios where keys can become compromised.

1. A hacker could get full and unrestricted access to the database holding all site keys and userid hashes.

   The hacker can then start cracking the $\texttt{AES}_{256}$ encryption by capturing all login traffic to the site for a long period of time. Since there is just a single block of 128 bits used for each login, which consists purely of random data, there is not much to go on.

2. A hacker could obtain the keyring from a user.

   The hacker would also need to know with which key(s) the keyring is encrypted. Should the hacker know that too, it would still need to know which key is used for what. If the hacker should know a key, then he or she can login to that site. He or she has obtained a single login.

   The site name is often entered using the keyboard. If possible, selecting a key from the keyring should be done using a graphical interface, so a simple key-logging Trojan would not be usable to gain knowledge about which key is used for which site.

3. A hacker could steal the backup copies of the Secret keys from a vault.

   With just the Secret keys compromised, no harm is done. Only when the database is also available, the hacker has full control. This full control is just for that site, and no other site is in any way compromised as well.

4. A hacker could break the SSL/TLS encryption of the HTTPS connection and view all traffic between user and website in plain text.

   The hacker would learn nothing, except which $\texttt{SHA}_{256}$hash code the user uses for that site. Using the hash to login requires a special replacement login program, to insert the hash directly in the data stream. It is hard to find a login that produces the same hash.

   All other login data is random and encrypted at that.

3 Storing user keys in a keyring

Keys for the user are collected and stored in a keyring. The keyring is a block of at least 100 random numbers with 128 bits, most of them dummy keys.

3.1 Creating a keyring

A keyring is created by using a random generator of sufficient quality, which generates 100 random numbers of 128 bits. These random numbers are then written to a file on a storage device. Key number zero is used to identify the keyring and will never change after its creation. The other 99 keys are dummies (random bits with no meaning whatsoever).

Nothing else is stored in a keyring, as to minimize the information stored there. Since all bits are entirely random—for dummies and real keys—a keyring stores no information whatsoever.

3.2 Encrypting a keyring

Optionally but preferably, the keyring on the storage device can be encrypted, to further enhance security. The unencrypted values should never be written to a storage device and only be kept in volatile memory. An encrypted keyring is of no value to a hacker without knowledge with which keys to decrypt it. It is up to the user to remember this.

The keyring can be encrypted by using the keys in the keyring itself. The user may select any number of different random key numbers from its ring, which are numbered from 1 to 99. Suppose two keys are used. Then, prepending a zero for key numbers less than 10, this yields a 4 digit PIN code for the keyring.

Using the keys (in selected order) to encrypt the keyring, the values of the keys are obscured. Select the first key from the keyring by specifying its index j. For each key $K[i]$ (with i running from 0 to 99), except $K[j]$, use XOR:

$$K[i] = K[i] \oplus K[j]$$

To decrypt, reverse the order of the selected keys and do the same. This will render the original keyring.

3.3 Copying keys and keyrings

A keyring can be freely copied to other devices, so all keys are available there as well. The keys of one keyring can be copied to other keyrings, in different locations. It is up to the user to keep a registration of this.

Encrypted keyrings can be stored in a public place, for easy access, and act as a backup. A dedicated webserver can be employed for storing encrypted keyrings, which can be used to restore lost keyrings. They can also be used to login when away from home with no access to your own keyring. These temporary keyrings should be discarded when logging out.

3.4 Irretrievably lost keys and keyrings

When a keynumber is forgotten and no record of it can be found, the key must be considered lost.

As the user has supplied websites with personal information, it may be easy to regain login capabilities by just asking for a new key, provided a username and maybe some other information can be given. Selecting a free keynumber on the keyring and replacing that key with the new one should restore the ability to login.

Keyrings can become unusable or lost in two situations: the device holding it is defective (like a harddisk crash) and no backup has been made, or the encryption cannot be reversed (PIN forgotten). In both cases, the user has to start over and create a new keyring, containing only dummy keys.

4 Logging in

The user, wanting to login to a site, has to lookup or remember the keyring's PIN (if any), the key number, and the userid. Having unencrypted its keyring, selected a key number (k), and entered the userid, the login process can commence. For logging in, the user needs a little program to do some calculations and to operate on the keyring.

4.1 Applying for a userid

When a new user applies for an account, it should choose a userid—one that the website will accept as unique. Furthermore, it should select the index of a hitherto unused key (therefore, a dummy). The user will send its $\texttt{SHA}_{256}$ hash of its userid, along with its dummy key K_d to the website.

The website will send the dummy key K_d to an SDPS hosting an HSM (see section 5.3), which will do the following:

1. Generate a random
$$K_s = \texttt{Random}()$$
which will be the new site key for this user.

2. Then it will encrypt the K_s with the current Secret key $S[0]$ to yield the new user key K_u
$$K_u = \texttt{AES}_{256}(K_s, S[0])$$

3. Finally, the new user key is encrypted with the user's dummy key K_d with an XOR operation:
$$K_x = K_u \oplus K_d$$

The SDPS will then send the new key K_s and the $\texttt{SHA}_{256}$ hash of the userid to the server hosting the accounts database. It will send K_x in a mail to the new user (optionally with the index the user supplied).

The user's keyring program can import the new key K_x into the keyring by calculating
$$K_u = K_x \oplus K_d$$
using the index it has originally selected for the slot in the keyring. The dummy key K_d is overwritten with the new key K_u.

4.2 Basic login scheme

The procedure described below to login is the basic login scheme.

4.2.1 Step 1: knock, knock...

The user's login program has to compute the folowing:

1. A random number (R_u) the website will need to appreciate by returning its hash later on:
$$R_u = \texttt{Random}()$$
2. The user key (K_u) is taken from the keyring at index k and the random number is encrypted with it to get the value A_u:
$$A_u = R_u \oplus K_u$$
3. The website will return a hash value of the user's random value. We need to be able to verify this hash, so we compute our own hash (B_u) to verify it with:
$$B_u = \texttt{SHA}_{256}(R_u) \textbf{ mod } 2^{128}$$
4. Finally, we need the hash of the userid (U) to present to the website:
$$U = \texttt{SHA}_{256}(userid)$$

The hashed userid U and the encrypted random number A_u are sent to the website.

4.2.2 Step 2: site key lookup

After receiving U from the user, the site key K_s needs to be looked up. A query with U is sent to the database with account information. If a match is found, K_s is returned; otherwise, K_s is set to zero, indicating that no such record exists.

In the latter case, the values B_s and P_s are both set to zero as well, and returned to the user. The user needs to rethink its actions and start over.

4.2.3 Step 3: SDPS actions

Now an iterative process starts, trying to find the right Secret key to log the user in (see also section 2.1). The loop runs at most $min(m, \texttt{MAX\_ACTIVE\_KEYS})$ times. The website will send three values to the SDPS (see section 5.3): A_u, K_s, and a Secret key index i, starting at 0.

In case i is less than m (the number of stored Secret keys, see section 2.1) the HSM calculates the following:

1. Temporarily, regenerate a user key, using the same algorithm as when the key was originally generated:
$$K_u = \texttt{AES}_{256}(K_s, S[i])$$
with $S[i]$ the i-th Secret key stored in the HSM.
2. With the user key K_u, decrypt the random the user has sent:
$$R_u = A_u \oplus K_u$$

3. Calculate a hash with which the user can verify we own the site key K_s and a corresponding Secret key $S[i]$:

$$B_s = \texttt{SHA}_{256}(R_u) \textbf{ mod } 2^{128}$$

4. To be able to verify that the user ows and knows its user key K_u, is not sending a replay of some earlier succesfull login sequence, and will be unable to ly about the correctness of the hash of the random the site will send, we calculate a challenge for the user.

   If the index i equals zero, generate a random value

$$R_s = \texttt{Random}()$$

   otherwise, compute

$$R_s = \texttt{AES}_{256}(K_s, S[0])$$

   which will be the new key for the user.

   We then encrypt R_s to a value P_s the user can decrypt using its key:

$$P_s = R_s \oplus K_u$$

   This value will be different for each time the loop is executed, even if the same new key is transmitted, since K_u will change each time.

5. Calculate the expected response also:

$$Q_s = \texttt{SHA}_{256}(R_s) \textbf{ mod } 2^{128}$$

The HSM will produce the values B_s, P_s, and Q_s as a result of the calculations and the SPDS will send them back to the site, as an answer to the three values it was given (A_u, K_s, and i). The HSM will delete all temporary values from memory directly afterwards, and remember only its Secret keys.

Should index i equal m, then the array of Secret keys is exhausted. If we reach this situation, we cannot log the user in since all possible attempts have failed. Return zero values for B_s, P_s, and Q_s to indicate this.

4.2.4 Step 4: user verifies site key

The site sends B_s and P_s to the user. If both are zero, the login has failed and the user should return to step 1. It is advisable for the user to choose different values for the next try.

If B_s contains a nonzero value, the user verifies if B_s equals B_u.

If the two values do not match, the user either has selected the wrong key or uses an old key. A key can be old if it comes from a keyring that was not used recently. Another possible reason for a key to be old is when the Secret key of the site has changed recently.

To indicate that no match has been found, send

$$Q_u = \texttt{0x0}$$

(128 zeroes) to the website. The SDPS will need to start over, now with values A_u, K_s, and $i + 1$. So, back to step 3.

4.2.5 Step 5: site verifies user key

If B_s matches B_u, then the website has found a Secret key for the user. The user can now prove it has control over its user key by computing

$$R_s = P_s \oplus K_u$$

and

$$Q_u = \texttt{SHA}_{256}(R_s) \textbf{ mod } 2^{128}$$

which is sent to the website as proof. If the website accepts Q_u as correct it will log the user in.

If the website receives a value Q_u that does *not* match, something fishy is going on. Further attempts for this userid, or from that IP address should be scrutinized.

4.2.6 Step 6: key replacement

If this attempt to login was not the first with this key, the website has sent the user a new key in R_s (instead of a random value). The user should store this new key in the keyring, overwriting the old key the user has just used.

4.3 Optimizations and alternatives

With the basic login scheme several optimizations and alternatives are available. Here are some obvious ones.

4.3.1 Aborting the login procedure

Logging in with an old key every now and then is inherent in this scheme, as most Secret keys will be changed at regular intervals. Therefore, it is likely to sometimes receive a hash value B_s that does not match B_u. An unsuccessful second attempt may be the result of using a seldomly used key or a wrong key. So, after at least two unsuccesful attempts the user may be presented a question whether it likes to select a different key from the ring, or continue trying with this key. If the user opts to select a different key, the login procedure has to be aborted. This may be done by setting

$$Q_u = \texttt{0xffffffffffffffffffffffffffffffff}$$

(128 ones) and send this to the website (instead of $0x0$). We return to step 1.

4.3.2 Detection of false logins

If someone has stolen a keyring it should be useless. Sequentially using all keys in the keyring to try to find the right key should not work. After each unsuccessful attempt the account should be blocked for exponentially increasing amounts of time. Starting from, say, a second, but after a second attempt a minute, then five minutes, an hour, a day, a week.

4.3.3 Using last login date

Instead of always starting with Secret key $S[0]$, the date of the last login of the user can be taken into account. If the query with U as key that is sent to the accounts database would also return the last login date a starting key could be selected. It is no use trying new keys when you know beforehand that these keys were added after the last successful login.

4.3.4 Conditional key replacement

For most websites, restoring user keys that refer to the most recent Secret key $S[0]$ will be done without hesitation. For some, refreshing keys will be done only when certain conditions are met, like payment of monthly fees, a certain number of reviews written, or some amount of data uploaded. Until then, logging in with a valid (but in this context a typically 'old') key is granted. But if, for instance, payment is overdue, the key will expire and logging in is no longer possible.

To indicate that no new key is sent, use

$$P_s = A_u$$

and return this value to the user. In this case, no keys should be decrypted or overwritten.

4.4 Changing Secret keys

At regular intervals a new Secret key is introduced and, at the same time, an old Secret key may be sent to the Eternal Hunting Grounds.

4.4.1 Terminated accounts

When deleting the oldest Secret key from memory, all accounts with a 'last login' date older than the installation date of the second oldest Secret key should be marked 'terminated'. Removal of the oldest key renders all those keys unusable. Those accounts can be purged, as they cannot be used again.

4.4.2 Expired accounts

In some situations user access must be barred until some condition is met. Accounts can be made temporarily inaccessible for this purpose by letting keys expire. Expiration can happen automatically when users do not login in time to get their keys replaced, or intently by denying key updates. Expired user keys are associated with Secret keys with an index in the range $[\texttt{MAX\_ACTIVE\_KEYS}, \texttt{MAX\_KEYS})$.

Expired accounts can easily be made active again by using inactive Secret keys for the login process (and then renew the key).

Expired accounts can be reinstated, but only before the associated Secret key is erased from memory. After that, there is no way to regenerate the user key for logging in. The user should apply for a new account if it wants to regain access.

5 Required hardware

Apart from a server to host the website itself, you need a Secure Device Provisioning Server (SDPS) that contains a Hardware Security Module (HSM), which stores an array of Secret Keys and performs cryptographic functions. Yet another server should host the accounts database, linking at least userids with site keys.

5.1 Firewall

There should be a firewall between the webserver and the SDPS, to protect the SDPS from attack should the website be compromised. The SDPS should be placed in a

DMZ. The other link from the firewall will be to the server that hosts the accounts database.

5.2 Accounts database

There should be a server for the database with account information. This database will probably be used for many other things that the website needs as well.

5.3 SDPS

The Secure Device Provisioning Server (SDPS) has a Hardware Security Module (HSM) for storing cryptographic keys, performing cryptographic functions, and generating random numbers. It should be separated from the rest of the website's internal network by a firewall, and placed in a Demilitarized Zone (DMZ) to protect it from misuse or attack.

The data that is sent to the SDPS, and the data that is received back from the SDPS are encrypted by design. Securing this channel with extra encryption is not really needed, as network sniffing yields only random values.

As such, a single SDPS can be shared among websites or multiple hosts for one large website, and its data sent over the Internet.

For redundancy, multiple SDPS systems should be employed, for if an SDPS fails, no one can login. When changing Secret keys in this scenario the keys need to be replaced simultaneously. When a user got a new key from an SDPS with the new key, and logs in again, an SDPS that misses the newest Secret key cannot verify the user any more.

5.4 HSM

The Hardware Security Module is a piece of tamper proof hardware. It stores cryptographic keys, certificates, and other data in a secure manner. It also has a whealth of cryptographic functions and can generate good quality random numbers. The HSM should be programmed in such a manner that even the SDPS, which hosts the HSM, is only able to use a small set of functions.

Ideally, the set of functions the SDPS should provide should be programmed directly into the HSM. This way, the Secret keys can be kept secret, and the SDPS cannot be misused by a hacker to regenerate login data.

6 Software

Several little pieces of software are needed to make things happen.

6.1 SDPS functions

The SDPS (see section 5.3) needs to perform the following functions (in relatively random order):

- Since the SDPS hosts an HSM, it could offer to generate a new keyring for a user (just a blob of 100 random 128 bit numbers).

- Generate a set of keys for a new user (K_s and K_x, see section 4.1) given a dummy key K_d and a $\texttt{SHA}_{256}$hash of a userid.

  It should send the site key K_s and the hash to the server that has all account information in a database using an insert query. The encrypted user key K_x should be sent back to the website, so it can transport it to the user.

- Store a new Secret key, given the new key and `MAX_KEYS`; the latter may change, due to a policy change.

  The new key should be stored in slot 0 of the array, by first moving all existing keys one position. If there are already `MAX_KEYS` Secret keys stored in the array, all keys with an index greater or equal to `MAX_KEYS` should be discarded.

- Calculate the values B_s, P_s, and Q_s, in response to the three values A_u, K_s, and an index i (see section 4.2.3).

  The input values come from the website; the output values should be returned.

6.2 HSM

The HSM in the SDPS should just present the functions needed for the SDPS, and nothing more. This way, misuse can be prevented. Anyway, it should be able to perform the functions described in section 6.1.

6.3 Accounts Database

The database that is used for holding all accounts should be able to return the site key K_s in response to a $\texttt{SHA}_{256}$hash of a userid the user has sent (see section 4.2.2).

6.4 Website

The website should be able to do some basic tasks:

- Relay values received by the user to the accounts database and accept the site key K_s (see section 4.2.2).
- Iteratively send values A_u, K_s, and an index i, to the SDPS, and relay the results back to the user (see section 4.2.3).
- Act according the replies of the user to find the right Secret key (see section 4.2.4).
- Verify that the user has sent the right hash Q_u and log the user in (see section 4.2.5).

6.5 Browser

The software for the browser has to do some diverse tasks:

- Present all keys in some graphical form, so the user can select one.
- Compute the $\texttt{SHA}_{256}$hash of the userid, generate a random value, XOR this with the key selected by the user, and send this to the website (see section 4.2.1).
- Verify the $\texttt{SHA}_{256}$hashes the website will send (see section 4.2.4).

- Decrypt the challenge from the website, compute the $\texttt{SHA}_{256}$hash of the result, and send it back (see section 4.2.5).
- Optionally, replace keys on the keyring (see section 4.2.6).

7 Conclusion

The security of the login process for websites can be greatly improved by using a keyring at the user side and an HSM at the website's side. Instead of sending relatively short, easy to guess strings (passwords) over the line, the use of encrypted random values, of 128 bits each, is a big improvement. No keys are sent, just random values, which will be different each time a user logs in.

For hackers, getting login data in huge numbers will be very difficult, since this data is no longer stored centrally, but split between website and user. Each part alone has no value, and all values stored at the website render no valid login data without Secret keys. These keys are kept in very secure hardware: an HSM. Each and every user key is encrypted with different other keys on keyrings. Sniffing network traffic (even on links without SSL/TLS encryption), or collecting keystrokes with Trojans will not help. Keyrings are encrypted, so to no direct use to hackers as well; they may be stored on the web for easy access and backup.

Several important security measures are automatically implemented: keys are changed frequently (as frequent as Secret keys are changed), they differ for each website, and keys on a keyring and the knowledge which key is used for what site constitutes two-factor authentication.

For the user, the way to logon to websites will change, but it will be an improvement over the burden of keeping track of all passwords. A single key number for a website is all you have to remember.

The software to make it all happen is easy to write, since the algorithm for logging in, and to handle keyrings is not difficult. No hardware on the user's side is necessary. The hardware to implement this on the website's side is relatively cheap, compared to all other security measures a website (still) needs to have (Firewalls, Intrusion Prevention Systems, etcetera).

References

[1] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, December 1999.

[2] Bruce Schneier. Write Down Your Password. Cryptogram Newsletter, June 2005.

[3] Wikipedia. 2012 linkedin hack — wikipedia, the free encyclopedia, 2012. [Online; accessed 17-September-2012].