Secure Login without Passwords

T.M.C. Ruiter, $X{\oplus}R_{\mathsf{key}}$

February 8, 2014

Abstract

These are the ingredients for secure logins:

- For login purposes, the webserver uses a separate login server. This server can be internal (in the back-office) or external (somewhere on the Internet).
- The login server 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 for the login procedure are performed by the login server and take place inside the HSM; no values are remembered afterwards, except Secret keys.
- For each user, a random number acts as the site key for that user. The user gets a different 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 $\texttt{SHA}_{256}$ hashes of random 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, most of them dummies. No other information whatsoever is stored in a keyring, which may be stored in an encrypted file.
- 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.

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 ([2]), 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 [3].) 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 14 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. To see what I mean, watch [4].

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 A new login approach

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

This proposed new way of logging in has several advantages over the current practice of websites requiring passwords. 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 Computers and links

This proposed way of logging in concentrates on computing values and communication of the results. It is therefore appropriate at this point to elaborate a bit on who is communicating with whom and why.

1.3.1 Computer roles

Four computer roles can be distinguished when logging in.

Webserver This is the front-end server to which the user is communicating. From our point of view, it is the central system, acting as a hub. It communicates with three other servers: the accounts server, the login server, and of course the client computer. We consider it the most vulnerable to 'visits' with malicious intent, so it has been appointed the most simplests of tasks. It does not compute anything remotely valuable, it just compares values provided by others.

Accounts server This server is typically located in a network only accessible by the webserver. It stores all kind of user related account data, but no userid or passwords; it stores hashes and site keys instead. Like the webserver it does not compute anything either.

Login server This server uses an HSM which stores an array of secret keys per website it services. This server computes nothing on its own, it lets the HSM do all the work secretly, using some of its finest cryptographic functions. It receives login requests from the webserver and returns values that the webserver can compare or relay.

Client computer With this device the user communicates with the webserver. It generates random numbers and uses `XOR` and $\texttt{SHA}_{256}$ to compute the values exchanged with the webserver.

These roles can be played by as many computers as there are roles, but that need not always be the case. In fact, each and every role may be combined; a single computer may perform them all, for that matter.

The accounts server and the login server may be combined in a single server in the back-office. The webserver and the accounts server may be combined, with the login server somewhere on the Internet. The combination of webserver, accounts server, and login server is also valid; although this concentration of roles is not highly preferred.

1.3.2 Secure links

Traditionally, the communication channel between the user and the webserver is secured by means of TLS. The user sees the URL that starts with "https://", which means that the webserver has authenticated itself using a digital certificate. The browser will validate the supplied certificate using other trusted certificates. This security measure is sufficient but required.

The communication between the webserver and the accounts server should be conducted over a secure channel as well.

Finally, the communication between the webserver and the login server needs to be encrypted. For this, TLS is a sufficient security measure, although in this case mutual authentication is a must. The webserver needs to know it is talking to a legitimate and trusted login server, and the login server needs to know it is receiving valid requests. The certificate of the websever is used to select which array of secret keys to use, as the login server may service more than one webserver.

2 Keys

The keys used in this login scheme do not all have the same length. The Secret keys have 256 bits, for instance. The `RSA` operations use keys of 2048 bits. Keys stored in a keyring have 128 bits, but fewer bits may actually be used. The number of bits in such a key may differ among websites, and is denoted in this article as value n.

Three sort of keys are used: user keys (on a keyring), site keys (in a database), and Secret keys (in an array in an HSM). A user key is the result of encrypting the site key for that user with a Secret key.

2.1 Secret keys

The keys used to encrypt other keys—called Secret keys in this article—are used to perform $\texttt{AES}_{256}$ encryptions of site keys to get user keys, and are 256 bits long. They reside in the HSM of a login server.

2.1.1 Array of Secret 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.

If there were only one Secret key, changing it would render all user keys permanently useless. Therefore, an array of Secret keys needs to be kept (with room for at least one old Secret key), allowing users to login using an old(er) key. Several Secret keys are used this way and are considered to be stored in array S, in this article.

1	2	...	MAX_ACTIVE_KEYS	...	MAX_KEYS

With this array, three values are defined.

1. The number of stored Secret keys is represented by m.

2. The constant[1] 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 up one position 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 youngest 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 $[1, \texttt{MAX\_KEYS}]$.

   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.

[1]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.

Upon successful login with an old key, the user will be provided with a new user key automatically, with which it can login using the newest Secret key, from that moment on. Changing of user keys is seamless (see section 2.3) 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.

2.1.2 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.

2.1.3 Expired accounts

In some situations user access may be barred until some condition is met (see also 3.3.6 on page 13). 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 [`MAX_ACTIVE_KEYS`, `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.

2.2 Site keys

For each user, a site key is generated once by a pseudo-random generator of the HSM of the login server (see section 3.2.3 on page 9), and need never be replaced.

The site keys are stored in a table of a database of the accounts server, where a hash value will act as the primary key for that table. Each time a user tries to login, the database is queried with the hash value the user supplies, and the site key for that user is returned for further processing.

2.3 User keys and keyrings

Together with the site key, a user key is computed by encrypting the site key for the user with the youngest (or current) Secret key, using the $\texttt{AES}_{256}$ algorithm (also see section 3.2.3 on page 9).

The keys users get from the website are stored on a keyring (see section 2.3.1 on the following page). The keyring will be considered an array Z in the rest of the article. A key in a keyring is automatically replaced with a new key by the webserver whenever the Secret key of the login server is changed (see section 3.3.6 on page 13).

2.3.1 Storing user keys in a keyring

A keyring is created by using a random generator of sufficient quality, generating a set of random numbers of 128 bits, stored in a file. Key number zero is used to identify the keyring and will never change after its creation. The other keys are dummies (random bits with no meaning whatsoever), some of which will be overwritten with real keys over time. Real keys in the keyring should optimally be put randomly between the dummy keys.

Typically, 100 random numbers are generated this way, so all keys are conveniently numbered with a one or two digit number (1 through 99), which can easily be remembered. However, you can put any number of keys in a keyring file. From zero keys (which leaves only the keyring identifier $Z[0]$, and allows you to login to zero websites) up to any amount of keys you bother to carry around with you.

Having more keys than sites you want to login to is just a way to make things a bit harder for those that do not own the keyring to choose keys. If you are not happy with this, you can just add new keys as you acquire them, just as with real keys.

2.3.2 Copying keyrings

A keyring can be freely copied to other devices, so all keys are available there as well.

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.

2.3.3 Irretrievably lost keys and keyrings

When a key number is forgotten, or a key inadvertently overwritten, and no record or backup 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.

2.4 Certificates and key pairs

As part of the security of the login process, keys need to be exchanged in encrypted form to and from the login server. Furthermore, the website may want to signify to its users that it is using a trusted login server.

To accomodate for this requirement, the login server needs a certificate that holds a public encryption key. The private key that belongs to it is stored in the HSM. With this certificate the users of the login system can validate the trustworthyness of the used login server by validating the chain of trust. The certificate is signed by a Certificate Authority (CA). This CA's certificate is also signed, etcetera, up to a Root CA. A list of all trusted Root CA's is stored in the browser of the user and is already used to validate the certificates of websites.

2.4.1 Login server keys

The login server uses two types of $\texttt{RSA}_{2048}$ keys: an encryption key and a signing key; both private parts tucked away in its HSM. The public counterparts of these keys are presented to the user with a certificate, signed by a Certificate Authority so the user can validate them.

The encryption keys of the login server are called K_{le} for the public key ('l' for 'login'; 'e' for 'encryption'), and K_{LE} for the private key. The signing keys of the login server are called K_{ls} for the public key ('s' for 'signing'), and K_{LS} for the private key.

2.4.2 Accounts server keys

Like the login server, the accounts server needs its own set of keys for encryption of other keys. No signing is neccesary, so we have K_{ae} as the public key ('a' for 'account') and K_{AE} as the private key.

3 Logging in

Having unencrypted its keyring, selected a key number (k), and given its means of identification, the login process can commence.

3.1 User Identification

Instead of sending a traditional alphanumeric userid, we send a combination of hashes. This value will be used by the webserver to identify the user, and the webserver will only store hashes in its user database. This value is specially crafted, so guessing will be hard.

The user will still need something as a means of identification. The most basic form of this is to give a name (real or imaginary); in this case in the form of a string of letters. Another form may include biometric features, like an iris scan, a fingerprint, or hand geometry. Or, in contrast, a hardware token may provide a certain value.

The identification value is chosen once per website and can remain the same as long as the keyring exists. There are no restrictions or constraints as to the contents of this identification value, apart from it to be empty. This means that it may be the same and reused for each and every website.

The hash that is sent to the website to identify the user, will be constructed using a combination of the keyring identifyer $Z[0]$ and the identification value (`ID`), and is constructed as follows. Compute the $\texttt{SHA}_{256}$ hash of the keyring identifier and the ID:

$$H_0 = \texttt{SHA}_{256}(Z[0])$$

$$H_1 = \texttt{SHA}_{256}(\texttt{ID})$$

Then, compute the final 256-bit value U_h by means of an `XOR` operation:

$$U_h = H_0 \oplus H_1$$

This hash value is a combination of something the user has, or has access to (the keyring) and something the user knows (its chosen name) or uniquely is (some biometric value) or phisically posesses (a hardware token). This makes it hard to match a stolen set of hashes from a webserver with a set of keyrings from a keyring server.

3.2 Applying for an account

3.2.1 Step 1: Excuse me...

When a new user applies for an account, apart from any personal details the website is interested in, it presents the webserver with its hash value U_h. Furthermore, the user should select the index (d) of a hitherto unused key and use n bits of it, at the discretion of, and indicated by, the webserver.

$$K_d = Z[d] \mathbin{\textbf{mod}} 2^n$$

Although this is a dummy key that is not used for logins (it will even be replaced after a successful application for an account) it still needs to be secured, since it is used in the application process. Eventually, this key needs to find its way to the login server, so we will encrypt it with its public encryption key K_{le}:

$$K_h = E_{\texttt{RSA}}(K_d, K_{le})$$

The user will then send U_h, K_h, and its age, shoe size, and gender to the webserver.

3.2.2 Step 2: Howdy partner!

The users' particulars and its identification hash value U_h are relayed to the accounts server, which will create the account. The webserver will request values for a new user from the login server, by forwarding the dummy key (K_h), and indicating how long the keys should be (n bits long).

3.2.3 Step 3: What have we here?

The login server will do the following:

1. Generate a n-bit pseudo-random site key K_s for the user, which is then encrypted with the public key K_{ae} of the accounts server:

   $$K_s = \texttt{Random}(n)$$

   $$K_y = E_{\texttt{RSA}}(K_s, K_{ae})$$

2. Encrypt the K_s with the current Secret key $S[0]$ to yield the new user key K_u.

   $$K_u = \texttt{AES}_{\texttt{256}}(K_s, S[0]) \ \mathbf{mod}\ 2^n$$

3. The (2048 bit) K_h value is decrypted with the private key K_{LE} to yield K_d:

   $$K_d = D_{\texttt{RSA}}(K_h, K_{LE})$$

   which is subsequently used to encrypt K_u:

   $$K_x = K_u \oplus K_d$$

   Both values K_x and K_y are returned to the webserver.

3.2.4 Step 4: Here are the results

The webserver will then relay the new encrypted key K_y and U_h to the accounts server. The accounts server will decrypt K_y with its private key K_{AE} to get the site key, but encrypt it again on the spot with the public key of the login server:

$$K_s = D_{\texttt{RSA}}(K_y, K_{AE})$$

$$K_a = E_{\texttt{RSA}}(K_s, K_{le})$$

and update the new account with it. These encrypted keys can be sent to the login server by the webserver during logins, without further processing.

3.2.5 Step 5: Thanks mate!

The encrypted user key K_x is presented to the user, so it can store it in its keyring. The user can decrypt its new key with by using K_d it used originally:

$$K_u = K_x \oplus K_d$$

Finally, K_u is imported in the keyring with something like

$$Z[d] = (\texttt{Random}(128 - n) << n) \oplus K_u$$

The dummy key K_d at $Z[d]$ is overwritten with 128 random bits, of which the last n bits contain the new key K_u.

3.3 Login scheme

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

3.3.1 Step 1: Knock, knock...

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

1. The userid hash U_h (see section 3.1).
2. An n-bit pseudo-random number R_u:

$$R_u = \texttt{Random}(n)$$

3. The user key K_u is taken from the keyring Z at index k. As this is 128-bit value, take the least significant n bits of it. The pseudo-random number is encrypted with it to get the value A_u:

$$K_u = Z[k] \textbf{ mod } 2^n$$

$$A_u = R_u \oplus K_u$$

4. The webserver will return a hash value of the user's pseudo-random value. To be able to verify this hash, we compute our own hash B_u to verify it with:

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

The special userid hash U_h and the encrypted pseudo-random number A_u are sent to the webserver.

3.3.2 Step 2: Site key lookup and key matching attempts

The webserver runs a login procedure.

After receiving U_h from the user, the encrypted site key K_a needs to be looked up. A query with U_h is sent by the webserver to the accounts server. If a match is found, K_a is returned; otherwise, K_a 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.

Along with K_a the account status s may also be returned. This status may indicate whether we want to allow the user to login or not. If something is required from the user (payment, or otherwise) we might want to expire the account until the requirement is met. We use the value `MAX_ACTIVE_KEYS` to limit the number of keys we want to try.

If `MAX_ACTIVE_KEYS` is set to 1 then accounts will expire immetiately when a new Secret key is inserted. When set to 3, there will be a grace period of 2 times the Secret keys replacement interval. This means that login is granted for this time, in which the user can fulfill its debts. If that does not happen, the account will expire.

Users that have not logged in for a while, but have payed their monthly fees, can still login without problems because all Secret keys will be tried for such logins.

Now, an iterative process starts, trying to find the right Secret key to log the user in. The webserver will send five values to the login server: A_u; K_a; the number of bits in the user and site keys (n); a boolean value indicating if the user is capable of performing a verify operation using a public signing key (v); and a Secret key index i. This will be repeated (increasing index i), until a match is found, or no more keys can or will be tried.

Finally, we could consider reducing the number of login attempts. Instead of always starting with index 0 for Secret key $S[0]$, the date of the last login of the user can be taken into account. If the query with U_h as key, that is sent to the accounts database, would also return the last login date, a starting key index i could be computed.

3.3.3 Step 3: Login server actions

The login server is a processor of login values and calls a function of the HSM to compute them .

In case i is less than m (the number of stored Secret keys, see section 2.1 on page 4) the login server calculates the following, all within the HSM:

1. Decrypt the site key, using the private encryption key K_{LE}:

$$K_s = D_{\texttt{RSA}}(K_a, K_{LE}) \textbf{ mod } 2^n$$

2. Temporarily, regenerate a user key, using the same algorithm as when the key was originally generated:

$$K_u = \texttt{AES}_{\texttt{256}}(K_s, S[i]) \textbf{ mod } 2^n$$

   with $S[i]$ the i-th Secret key stored in the HSM.

3. With the user key K_u, decrypt the pseudo-random the user has sent:

$$R_u = A_u \oplus K_u$$

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

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

5. If the user is able and wants to verify the signature of the login server, then the value B_s will be replaced with a signature thereof with the signing key K_{LS}:

$$B_s = S_{\texttt{RSA}}(B_s, K_{LS})$$

   We use and send only the signature, not the B_s value also.

6. To be able to verify that the user owns and knows its user key K_u, is not sending a replay of some earlier successfull login sequence, and will be unable to ly about the correctness of the hash of the pseudo-random the webserver will send, we calculate a challenge for the user.

   If the index i equals zero, generate a pseudo-random value

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

   otherwise, compute

$$R_s = \texttt{AES}_{\texttt{256}}(K_s, S[0]) \textbf{ mod } 2^n$$

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. This is because K_u will change each time, as it is the encryption of K_s with a different secret key $S[i]$.

7. We now know both random values. If the user knows them also (by successfully decrypting P_s) it can prove this by sending the `XOR` of both values:

$$Q_s = R_s \oplus R_u$$

This is the value that the webserver should compare.

The HSM will produce the values B_s, P_s, and Q_s as a result of the calculations and the login server will send them back to the webserver, as an answer to the five values it was given (A_u, K_a, n, v, 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.

3.3.4 Step 4: User verifies site key

The webserver 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 is nonzero, the user verifies whether this value matches with B_u.

If the user is not able to verify a digital signature, just compare B_u and B_s. Otherwise, it should try to verify the combination of B_u (the hash over R_u) and the signature thereof, which is in B_s:

$$V_{\texttt{RSA}}(B_u + B_s, K_{ls})$$

If the two values do not match, the user either has selected the wrong key or uses an old key. 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.

To indicate that no match has been found, we send

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

(n zeroes) to the webserver. The webserver will need to start over, and send values A_u, K_a, n, v, and $i+1$ to the login server. So, back to step 3.

After two unsuccesful attempts the user may be presented a question whether it likes to abort the login procedure or continue trying with this key. To abort, use:

$$Q_u = \texttt{0x1}$$

and send this to the webserver (instead of `0x0`). We return to step 1.

3.3.5 Step 5: Site verifies user key

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

$$R_s = P_s \oplus K_u$$

$$Q_u = R_s \oplus R_u$$

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

If the webserver receives a value Q_u that does *not* match, something fishy is going on. Further attempts for this account, or from that source should be scrutinized.

3.3.6 Step 6: User key replacement

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

$$Z[k] = (\texttt{Random}(128 - n) << n) \oplus R_s$$

where all bits of $Z[k]$ are replaced.

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.

Instead of incrementing index i in step 2, the index is decremented. The HSM will just be sending random values for P_s in that case, for all values of index i. To indicate to the user that no new key is sent, the website will replace it with A_u

$$P_s = A_u$$

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

4 Conclusions

The security of the login process for websites can be greatly improved by using a keyring at the user side and an HSM employed by the website. Instead of sending relatively short, easy to guess strings (passwords) over the line, the use of encrypted random values, up to 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. Sniffing network traffic or collecting keystrokes with Trojans will not help. Keyrings have very high entropy and 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. One userid for all sites and a single key number for a website is all you have to remember.

4.1 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 [5].

  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 them all.*

- 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.

- People are sometimes called by other people, claiming to be employees of banks. In order to "help" solve a problem, users are asked to give their login credentials. Some ignorent users are willing to oblige.

  With a keyring the only thing slightly useful to a hacker would be the userid. The keynumber is of no use, since the hacker has no access to the keyring itself; telling him which key index is used to login has no value.

Acknowledgements

Thanks to Martijn Donders for his cryptographic support, and Rob Bloemer for reviewing the first draft of this article. The review sessions with Jannes Smitskamp were pleaseant and intense, and helped a lot to expose some hidden flaws; which were all remedied elegantly.

References

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

[2] Mat Honan. Kill the password: Why a string of characters can't protect us anymore. *Wired Magazine*, 11 2012.

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

[4] Toby Turner. Password rant, December 2012. http://www.youtube.com/watch?v=jQ7DBG3ISRY.

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