EMCSSL

Decentralized identity management, passwordless logins, and client SSL certificates using Emercoin NVS.

Updated 19 May 2015

Emercoin International Development Group

http://emercoin.com

EMCSSL - decentralized identity management, passwordless logins, and client SSL certificates using Emercoin NVS.

This document will introduce details of a scalable infrastructure for passwordless authorization suitable for an unlimited number of web services.

The infrastructure sits on the Emercoin cryptocurrency blockchain, using the blockchain as a decentralized trust store of hash sums for client SSL-certificates. Certificates can be generated by clients locally, without any central authority, and quickly replaced as needed. This makes the system effective both for scheduled replacement and rapid recall of compromised certificates.

The uniqueness of the proposal is in the complete decentralization of the system, i.e. the lack of a group of servers running under a single authorization (as used in the systems of Kerberos, OpenID, TeddyID and the like). As a result, it is not possible for EMCSSL to suffer system-wide service disruption either due to technical failure or malicious attack upon authorization servers. In addition, it is not possible for a user to have their accounts globally suspended at the whim of a single authority.

Also outlined is InfoCard - a decentralized distributed "business card" system that complements EMCSSL's passwordless logins by allowing website profiles to be automatically populated. InfoCard has the ability to organize information in a hierarchical structure, which can be useful for quick content updates to all cards within companies or other organizations.

The current status

In today's internet, the main way for a server to authenticate a client is a password system. The idea is that you have a secret password, created when first using a site, which confirms your identity on subsequent visits. While it looks simple on the surface, this system has a number of disadvantages:

- **Weak password.** Creating the password is normally the responsibility of the user, who often chooses a password that is too easy to guess, such as a simple dictionary word, or a dictionary word with a slight variation.
- **Complex passwords.** To avoid weak passwords, many sites force users to invent complicated passwords that contain a combination of characters, numbers, and symbols; even the password «mWxbq7LECJ7m4Gtu95L» would be considered weak on sites with strict password requirements, because it contains no symbols like @!$. As requirements are not standardized, a password may be rejected for any number of other reasons - e.g. the password above would be rejected by some sites because its length exceeds a 16 character limit.
- **Complexity of password management.** Since there is no uniform standard for passwords, it is impossible to create a universal password generator that could create passwords to suit all sites. As a result, a number of "Password Keeper" tools have been invented - where a password-protected database contains all the other passwords. In addition, many sites require passwords to be changed regularly, and after a password change, users worried about forgetting their new password, will simply write it down somewhere. Needless to say recording passwords in plain text is a direct path to the passwords being compromised.
- **The need for a unique password for each site.** When passwords are presented for login it is possible for them to be intercepted in transit, and in some cases, passwords can also be stolen as a result of a site hack. If the same password is used for different sites, such password leaks can be disastrous - the owner of the compromised password has to frantically recall every use of the password and urgently change passwords on every site - before the attacker does!
- **Recovering forgotten and lost passwords.** Surely many people have encountered the procedure of trying to prove their identity to a site in the event of a lost or forgotten password, where failing to complete the process has resulted in the loss of the account.

There is an alternative

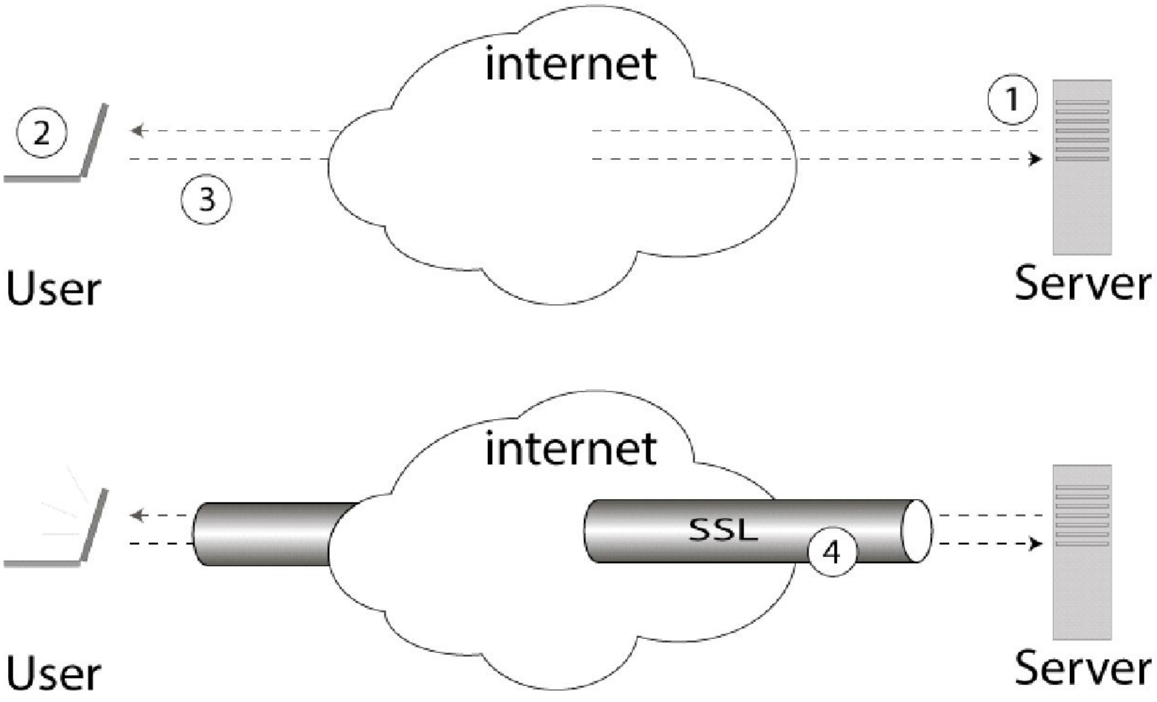
There exist a number of alternative solutions. In the corporate sector there's the popular RSA Token - a special device that generates one-time passwords every few minutes. These passwords must be entered on the website of the relevant provider.

Also, there are client SSL-certificates for browsers where a user buys a certificate signed by a trusted authority (who charges an initial fee, annual renewal fee, and additional fees for recall/replacement in the event of a compromise). Although the technology is effective, the significant price, constant dependence on the Certificate Authority (CA), and complexity of generating the certificate all make traditional client SSL-certificates impractical for widespread use. Indeed, today client SSL-certificates see only limited use in corporate networks while among the broad masses the technology remains virtually unknown.

There is also an option where authorization for a site uses client certificates issued by the same site, which are only recognized by them. There is no financial cost in this model, but the customer is required to perform the ongoing job of maintaining their certificate database. If a user has accounts on 52 sites (not uncommon for many internet users), and uses standard certificates for a period of 1 year, they would need to update a certificate on average once per week. Also, this certification procedure is not standardized, making it necessary to work within different requirements for each site. Therefore, despite its technical perfection in security practice, the last option has not been widely adopted and is unlikely to become so in future.

SSL as currently used

Currently, the most widespread mechanism for protection of a network connection is SSL - Secure Socket Layer, a mechanism whereby a server proves to the client that it is itself, and not a fraudulent duplicate server. In addition to the server proving its authenticity, SSL establishes an encrypted connection between the client and the server. Here's how it works:



A typical installation of a secure SSL-connection:

1 - The server sends its own SSL-certificate containing the server's public key.

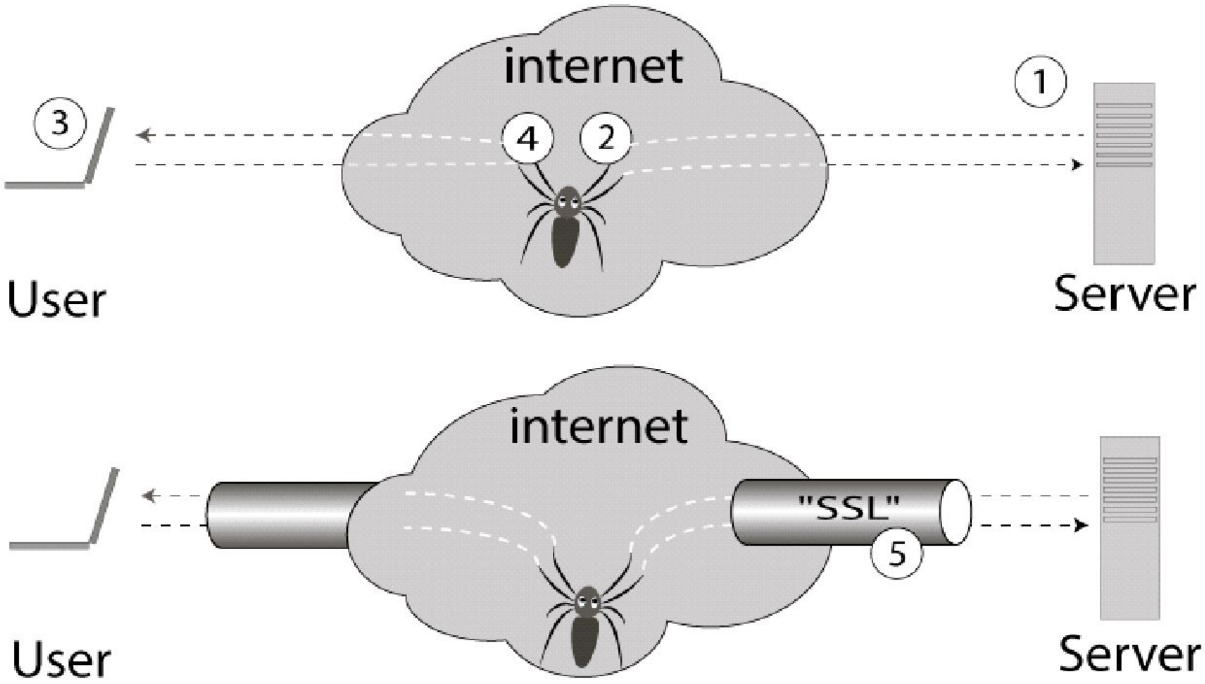
2 - The user verifies the signature on the certificate and verifies that the server certificate was issued by a trusted agent.

3 - The user sends a random number encrypted with the server's public key.

4 - In the event of successful decoding of the number on the server, a secure SSL session is established.

However, secure connections can be broken if an attacker were able to add themselves to the list of "trusted agents" for the user's browser, through the use of a "fake" browser root certificate, e.g. in a corporate network where the administrator can add certificates to the list of "trusted agents" and then organize a "man in the middle" attack in the https-connection.

The example below illustrates this attack:



Problem with SSL connections.

If a false root X.509 certificate can be introduced to user devices (with the purpose of tracking, traffic control, etc), then there is the possibility of "man in the middle" attack.

1 - SSL server sends the public key to the user.

2 - A certificate attacker intercepts and substitutes their own.

3 - The user checks the fake certificate and the check is successful, since the attacker is already on the list of "trusted agents".

4 - The user encrypts a random number with the fake certificate and sends it to the supposed server, which is in fact the "man in the middle".

5 - The attacker successfully decrypts the number and establishes a secure connection to the user. At the same time, the attacker establishes a secure connection to the server using the real server certificate.

6 - The compromised SSL connection runs under the control of the attacker.

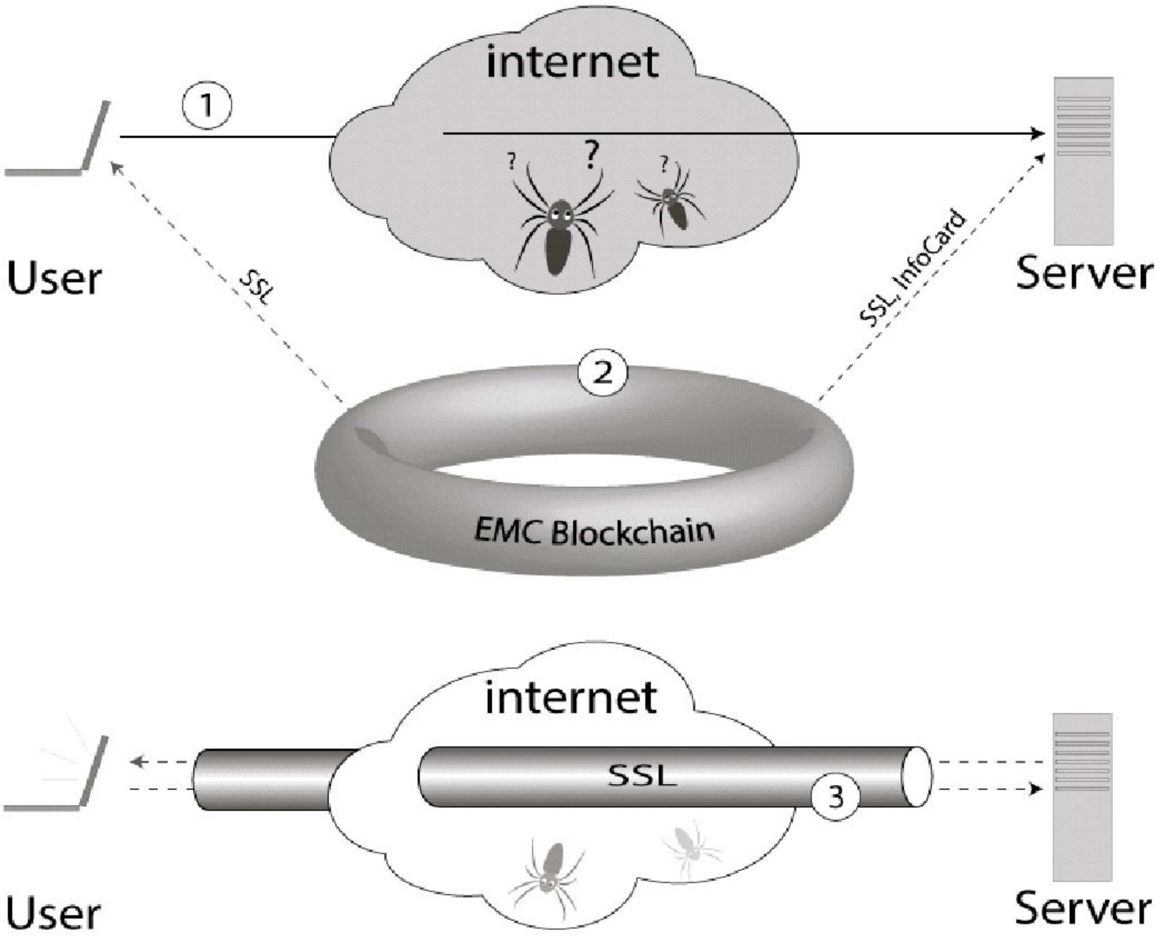
Introducing EMCSSL

EMCSSL is based on client SSL-certificates, but as well as client authentication, it also provides a secure encrypted channel of communication with the server, all in a single package. Unlike other SSL systems, there is no trusted CA - the role of CA is performed by the blockchain of the decentralized cryptocurrency Emercoin. Thus, the client SSL-certificates can be generated and updated completely on the client side without restrictions or need for interaction with another party.

In EMCSSL, the client SSL-certificate can be reused for authentication on multiple servers without sacrificing security. For normal internet use the user (you) would have just one single certificate, which radically simplifies the support of a large group of accounts, and eliminates the need for tens or hundreds of passwords. The option exists to use several certificates at a time, but this would not be needed to enhance security. Several certificates would only be needed if a user wished to maintain some "masks" i.e. different accounts.

In addition, EMCSSL does not permit "man in the middle" attacks as described above, since the server checks on the EMC blockchain whether it is a real certificate from the client, or fake (see below).

These differences combined produce for the first time a secure login authorization system that also has practical applicability for widespread use.



Preventing "man in the middle" attacks via blockchain (EMCSSL):

1 - The user sends their client SSL-Certificate to the server.

2 - Server authenticates the received certificate with the signature stored in Emercoin's distributed Key->Value (NVS) storage subsystem.

3 - Secure SSL session is established only when the certificate is authentic (no substitution is possible).

The user experience

1. You run a program locally to generate (or update) a private SSL-certificate.

2. You publish (or update) the public "digital signature" part of the certificate to the Emercoin NVS.

3. You upload the certificate to your browser.

4. It is only necessary to perform points 1-3 **one time**. After that, during the lifetime of the certificate (5 years by default but can be modified) you just do the following:

5. Visit any site that supports the system, and log in to your account without a user name and password. Everything is done automatically - as if you have a "perpetual login."

6. If you did not have an existing account on this site, you would simply press a button (something like "create new account based on the certificate") and an account is automatically created. Additionally, the account profile can be automatically populated with data that you provided when generating the certificate. You may choose not to provide additional details, but it is very nice when your account profile with a new site is automatically populated with one click.

How it works in practice

To generate client certificates with EMCSSL, download the toolkit for your operating system from https://pool.emercoin.com/emcssl, and unzip to any directory. The toolkit consists of three *.sh scripts for unix-type operating systems. The Windows-based version also has a set of win-bash openssl utilities, and *.bat files. No installation is required, you simply run the appropriate script. The site also contains a link to a test page that prints the contents of your EMCSSL certificate and InfoCard, if present.

Let's look at the generation and use of a certificate step by step:

1. Generation of a template.

First, you generate a certificate template with the help of the first program, **gen_tpl.sh**. The program asks the following attributes for the certificate:

- **CommonName** - username. This is required. You can provide a login name and username (separated by a space) or just a name.

- **Email** - your email address. This is optional. If you do specify, it can be used by the server to populate relevant fields.

- **UID** - optional. This is a link to more information about you (the owner of the certificate), if you choose to provide it. It can be used to link Emercoin InfoCard (NVS-service prefix 'info:'). The Emercoin InfoCard is stored in encrypted form within the blockchain and the info: link contains the decryption key. So information can only be accessed by sites where you use a certificate containing the link.

The program creates a template file containing the attributes you entered, using a random name to represent your certificate serial number and has the suffix * .tpl, e.g. **84aa5f2c6527eb33.tpl**.

This template can then be used to generate different certificates with the same parameters allowing you to easily update certificates when they expire, or if you suspect that the certificate has been compromised.

If desired, you can generate multiple templates each with multiple certificates, and then only update the latest as needed.

2. Certificate generation.

Next comes the actual generation (or update) of the certificate. To do this, we run the program **gen_crt.sh,** passing the template file to it as a parameter. Sample call:

```
./gen_crt.sh 84aa5f2c6527eb33.tpl.
```

The program generates a self-signed certificate (*.crt) and package for the browser (*.p12), containing the certificate and its associated private key. During this process, the program asks you to enter a password. This password will later be requested by the browser during upload package step (see paragraph 4 below). The password protects the contents of the *.p12 file from unauthorized use. Even if the file is stolen, the attacker can't use it without the password.

The private key that was used to generate the package is immediately deleted, as it has no further use. Following that, in addition to creating two files (*.crt, *.p12), the program will print an important message like:

```
Your new certificate is in the file 84aa5f2c6527eb33.p12

Please deposit into Emercoin NVS pair:

  Key:   ssl:84aa5f2c6527eb33

  Value: sha256=52cfd176a00756646fc73b6eecdf53c4d51cd077cc9b4ea49dd9bf660337fb52
```

First is the name of the freshly generated *.p12 file which you will install to your browser (in paragraph 4 below). Then there is the "key-> value" pair to be loaded into the Emercoin NVS. Important: the "key-> value" pair is printed only once and is not saved. You should use "Print Screen" if you want to capture it, but if it is lost, just generate a new certificate from the same template to obtain a new "key-> value" pair.

From this point, only the *.p12 file will be used. The *.crt file is a self-signed certificate left only "for reference". You can view it in a text editor and look at the structure and attributes of the certificate. If desired, the *.crt file can be safely deleted.

3. Loading the "Key->Value" into Emercoin NVS.

Using the GUI of the Emercoin wallet or the command line version **emercoind**, issue a name_new to register the abovementioned NVS pair into the blockchain (or name_update when replacing the certificate). Wait for the transaction to be confirmed in the blockchain (one confirmation is usually enough - taking an average of 10 minutes). As soon as you receive confirmation... congratulations, you have successfully registered your certificate in the distributed **EMCSSL** system and can proceed to the final stage! In the extremely unlikely event of a naming conflict (lower probability than winning millions in the lottery), repeat steps 1-3 again to generate a certificate with a different serial number.

4. Install *.p12 package into the browser and use.

After the successful registration of a certificate into Emercoin NVS, you can import the *.p12 file into a browser and start to take advantage of EMCSSL. Importing is no different from other SSL-certificate imports, and is done through the browser options. Here are guidelines for some common browsers:

Firefox: http://www.onlinehowto.net/install-ssl-certificate-in-firefox/784

Chrome: http://www.binarytides.com/client-side-ssl-certificates-firefox-chrome/

IE:
http://ipswitchmsg.force.com/kb/articles/FAQ/Using-client-SSL-certificates-in-Internet-Explorer-1307739573570

In general, you need to go to settings and find Advanced / Security / Certificates or similar, then click Import and upload your *.p12 file. In the process of importing you'll need to enter the password which you typed in step 2. After uploading the certificate, some browsers (e.g. Chrome) require a restart.

And you are done - now you can safely visit EMCSSL-enabled sites without separate passwords, and automatically propagate account information based on your certificates.

Let's examine how this works:

1. When you visit an EMCSSL-enabled site in the browser, the site requests that the browser present a client certificate. If the client has no certificate, or doesn't present one, the server, depending on the settings, can switch to a traditional password authentication system or refuse to proceed. But let's assume that a certificate exists, and that you are ready to submit it. The very first time you visit the site, the browser will ask, "Would you like to display this stored certificate?". You select the certificate, and the browser remembers that the server goes with that certificate, and will not ask the question again (the browser can be set to auto-select certificates, but this is not recommended).

2. Upon receiving a certificate, the server checks the certificate signature. Successful signature verification proves that the certificate was generated for the EMCSSL system, and nothing more. There is a difference here with the classic application of SSL-certificates, where we would certify the signature and the values of other fields of the certificate at this stage. But in our case, we are only checking that the certificate is generated for EMCSSL, and the remaining checks will be made later.

3. The server generates a random number (session password), encrypts it with the public key of the presented certificate, and sends it to your browser. The session password is established for this (and only this) connection.

4. The browser, with a complete *.p12 file, extracts the private key and uses it to decrypt the password sent by the server, to establish a secure https-connection with the server. Such a connection proves to the server that the bearer of the certificate owns the corresponding private key. Note that in this system, the private key never leaves your computer. Even if your certificate is intercepted while being transmitted over the network, no one but you can use it, because an attacker will not know the private key.

5. The server, after making sure that the client has a valid private key, checks the certificate against the information in the Emercoin blockchain. To do this, it extracts the certificate serial number and performs an EMC NVS search on this serial number to obtain the certificate hash that you uploaded to the blockchain. The server calculates a checksum for the newly received certificate against the corresponding serial number in the blockchain. In other words, the server confirms that the client's certificate, which contains the serial number N – is the same client who previously visited with the same serial number N, because this serial number can be registered only once within Emercoin's NVS subsystem. Indeed, if an attacker generates another certificate with the same serial number as yours, they wouldn't be able to upload the same checksum to the blockchain, as it is already taken by you. And if they generate a certificate with a different serial number - it will have a different UserID, and the server would create a separate account.

After all these checks (which occur in a fraction of a second), the server verifies that the client:

- Has a valid certificate in the EMCSSL system.
- Owns the private key.
- Owns the certificate serial number.

... and then gives you access to your account.

Note that the system verifies only the certificate serial number, which is considered the account's unique UserID. When next generating a certificate, you can populate the *.tpl with different field values for CN / Email / UID and then replace the certificate in the browser and amend the checksum in the corresponding NVS entry by issuing a name_update. After that, the old certificate will be useless, as it will fail the checks in paragraph 5 above.

This allows for quick and easy withdrawal and replacement of the certificate - upon suspicion of compromise of the certificate, or whenever you choose. You can always quickly generate a new certificate from the same template and amend the NVS values in the blockchain. Furthermore, none of this requires the participation of external certification authorities.

The server can extract CN (username) and Email fields from your certificate, and instantly fill in those fields when you create a new account on a website. Additionally, the certificate may contain a value in the UID (optional). This value represents a link to an **InfoCard** - additional information you wish to provide about yourself. The information is also stored in Emercoin's NVS, but encrypted and not publicly available. The UID field also contains the InfoCard decryption key, so access to information contained in an InfoCard is only given to a server if you have already given it your certificate.

When used together, EMCSSL and InfoCard allow the creation of a detailed and complete account profile, without entering a new password, filling in personal details, verifying email, etc - instead you simply click, and it's automatically done!

Also related to these Emercoin enhancements, is the interesting possibility of sending Emercoin currency (EMC) payments to a name in the NVS, i.e. the possibility of making secure payments. A server using this method, is guaranteed to send a payment to the owner of the certificate and no one else. Even in the event of an account hack, the withdrawal address cannot be substituted and funds cannot be sent somewhere other than the intended wallet. The introduction and use of this feature can greatly improve the security of cryptocurrency exchanges and mining pools, among other uses.

Frequently asked questions

- Is it possible to use the same * .p12 file in multiple browsers? And on different computers?

- Yes, it is!

- What if someone stole my laptop/tablet/phone which had a browser with a certificate?

- As quickly as possible you need to generate a new certificate, and update its digital signature in the Emercoin NVS. This step will automatically make the stolen certificate invalid. Fortunately in this case, it is only necessary to replace a single certificate to secure many accounts.

- What else can be done to protect the certificate from unauthorized use?

*- It should be on a PC or tablet that has a password, so even if stolen, the thief could not immediately access your browser. Additionally, some browsers (e.g. Firefox) have an option in the settings that requires you to enter the password for the certificate (which you entered when generating the *.p12 package), each time you use the certificate. It is also possible to store certificates in hardware cryptographic devices that use PKCS#11 format (http://en.wikipedia.org/wiki/PKCS_11), but unfortunately, not all browsers support such devices. In any case, your browser and control over the computer are the most likely sources of issues, rather than the network connection.*

- Still, if the attacker got into my browser - does this mean that without a password they can visit all my sites that are in the system?

- In general, yes. This system authorizes only the computer (or rather, the browser) and makes a secure network connection, but does not authorize the operator - it is believed that the controller of the browser is a valid operator. For authentication with critical services, sites should still use multi-factor authentication schemes e.g. three-factor - certificate, password, and a cell phone. EMCSSL is an effective alternative mechanism to storing passwords in the browser, and establishing a secure connection without the possibility of interception on the network, but it is not a silver bullet, and can not solve ***all*** *potential problems of information security.*

- Should an EMCSSL-enabled server still maintain a valid classic SSL (e.g. issued by a CA)?

Yes, because the client must still be sure they are connected to the right server, not a fake or phishing site (there is no sense to proceed with a secure connection with untrusted server).

- And why Emercoin? Can I use other cryptocurrency blockchains?

- You can, if the cryptocurrency has distributed storage, with a universal open protocol like Emercoin's NVS. The vast majority of cryptocurrencies do not have anything like this and those that do (e.g. Namecoin, NXT) have limitations that reduce functionality or security of the system.

[Worth mentioning is the «authorization token» system introduced by the NXT cryptocurrency (see http://nxtcoin.wikia.com/wiki/Nxt_Software). The idea here being that for each pair (site, username) a unique authentication token is generated for the client to present upon login, and the received token is checked via the NXT blockchain. Unfortunately, as the authorization token is sent whole, this method does not ensure security and makes it possible for the token to be intercepted in a "man in the middle" attack, or by phishing sites or similar, which would give the interceptor the token for their unlimited malicious use.]

- Is the system already being used anywhere? Can I see it in operation?

- Yes, you can! This system is currently used as one of the authentication mechanisms in the main Emercoin mining pool: https://pool.emercoin.com

- How do I access the Emercoin NVS service?

- First, download the free Emercoin wallet (http://sourceforge.net/projects/emercoin/files), buy a small amount of Emercoin from an exchange (or obtain from a faucet or giveaway, etc), and load your signature to the distributed database.

- And how much does it cost to upload a signature to Emercoin's NVS?

- It depends on several factors, but as a guide, around ¼ EMC to upload a single Key->Value pair. At the current rate 1EMC = 0.02USD, that's half a cent for 5 years (the life of the certificate), making it accessible even to those in very poor countries.

- Why not do it all free?

- Micropayments for the NVS service were introduced as a means to prevent misuse that would fill up the blockchain with "junk information." This amount is affordable and will prevent the blockchain being spammed with tens of gigabytes of useless data.

- Where can you buy EMC?

- See http://emercoin.com#services for a list of current exchanges.

You can purchase them from any holder, or mine it yourself on a pool such as http://pool.emercoin.com.

EMC can also be received as payment for assisting in the science project folding @ home: http://emerfor.org/folding_home/about_en.php.

- Does the EMC spent on uploading to Emercoin's NVS go to the developers wallet?

- EMC spent on NVS is 'burnt' i.e. permanently removed from circulation and not received by anyone.

- What duration of Key->Value lease time should I choose?

- We recommend a registration period of 10 years or more. It's worth noting that when you update the NVS record in future, the remaining lease time is not lost but added to. However, there is a direct analogy with domain names - if the NVS record is not renewed on time, it will expire and become available for someone else to register. Luckily, there is a simple recipe to avoid this: In the case you make the NVS lease time much greater than certificate lifespan, the NVS-record will expire long after the certificate, providing sufficient time to generate a new certificate and update the record after the certificate expires.

InfoCard

InfoCard - a data format by which people or organizations can choose to communicate some personal or business data. It aims to be structured a bit like a business card (paper or electronic *.vcf), or a user entry in a LDAP-directory.

InfoCard is another technology that uses the Emercoin NVS-subsystem. Generally speaking, it is a stand-alone system, however InfoCard's true value is seen when combined with EMCSSL, where a certificate's UID field contains a reference to an InfoCard record. When verifying a certificate, a server can extract InfoCard user information to fill in the user profile on the site, or contact the user with information contained in the InfoCard, including for cryptocurrency payments to an address indicated in the InfoCard.

Here we see an analogy with the OpenID system, which can authorize a user profile on a site using some other well-known site such as facebook. However, unlike OpenID, InfoCard is a decentralized system and does not depend on the operability of a single server. Assume a hypothetical situation where facebook, at their discretion, blocks your account - suddenly you'd no longer be able to post on dozens of forums, which had previously gone through facebook OpenID.

In addition, unlike the storage of details in systems such as OpenID, LDAP, vCard, where each record (card) contains **all** information about the person (or position), an InfoCard entry allows you to import information from other entries already in the system, providing convenience for employees of organizations or other hierarchical structures.

For example, the InfoCard system may contain a record for an organization (let's say a company), which includes the company name, address, phone, and other attributes, while the employee InfoCard records each contain a link to that information. Then if, for example, the company changed their phone number, or the office moved to another address, or merged with another company and changed their name - all of these changes would be made in the single InfoCard record for the company, and these changes would be reflected automatically in the cards of all the employees, so they will always contain the latest information. Remember how many times you were handed a business card with the phone number crossed out and the new one written in by hand!

Link to InfoCard-entry is as follows:

info:InfoCard_key:InfoCard_password

- *info* - NVS service prefix
- *InfoCard_key* - key to find the cards in NVS
- *InfoCard_password* - password decryption of the map

As mentioned above, all records in the InfoCard-subsystem are encrypted, and cannot be retrieved by looking directly at the blockchain. However, it is easy to retrieve records if you have the relevant decryption key.

The record structure in InfoCard is simple, and allows you to edit the file in a text editor.

Let us consider an example InfoCard (line numbers added for reference only):

1 Alias	johnnycitizen	# Short name (username, login)
2 FirstName	John	# First (short) name
3 LastName	Citizen	# Remaining part of full name
4 HomeAddress		
5	1313 Satoshi Lane	# Free form address
6	Level \# 1	# Free form address
7	Nakamoto Heights	# Free form address
8	Anytown, CA	# City
9	95555	# ZIP code
10	USA	# Country
11 HomePhone	+ 1-555-123-4567	
12 WorkPhone	+ 1-555-123-4568	
13 CellPhone	+ 1-555-123-4569	
14 Gender	M	
15 Birthdate	19910527	# May, 27, 1991
16 Email	jcitizen@bubbleinflators.com	
17 WEB	http://bubbleinflators.com/jcitizen	
18 Facebook	John.Citizen	
19 Twitter	JohnCitizen	
20 EMC	EdvJ7b7zPL6gj5f8VNfX6zmVcftb35sKX2	# Emercoin payment address
21 BTC	1MkKuU78bikC2ACLspofQZnNb6Vz9AP1Np	# BitCoin payment address

Each entry represents a set (in the mathematical sense) of pairs:

Key (one or more spaces) **Value**

Value - can contain spaces.

Example:

Alias johnnycitizen # Short name (username, login)

A single key may have a value composed of multiple lines of text. To add multiple lines to the desired key, the lines must begin with a space, that is, without giving a key. An example is HomeAddress in lines 4-10.

The "#" Character indicates the start of a comment which will be hidden when the card is being read. If you need to enter the symbol "#" inside a value, it's required to put backslash escape character "\" in front of it, as seen in line 6 of the example.

To import the contents of another InfoCard, the command would look like:

Import info:InfoCard_key:InfoCard_password

e.g. Import info:2f2c5a7c57d60668:74744c6e4443df490eab0807052bb9

Multiple imports are permitted, i.e. it's possible to include links to multiple other InfoCards, within the same InfoCard.

To prevent misuse and attacks on the system, the total number of imports in the generation of a response can not exceed 20. We believe that this enough for any conceivable hierarchical structure. And of course circular references are ignored.

Bear in mind when you type, that InfoCard data is processed from top to bottom, sequentially modifying the contents of the result set. So a sequence of instructions:

HomePhone +1-555-123-4567

HomePhone +1-555-123-0000

will create a unique value for HomePhone, +1-555-123-0000 (the previous value will be overwritten).

In order to correctly combine values, use the "+" symbol as qualifier before or after the key. Plus before the key means "add this value to the tail of the list", whereas a plus after the key means "add this value to the head of the list". Adding no qualifiers will delete values, and keep the last value, if any. This mechanism provides the flexibility to combine imported values. The examples below show these options:

HomePhone +1-555-123-4567

+HomePhone +1-555-123-0000

Result: HomePhone -> [+1-555-123-4567] [+1-555-123-0000]

HomePhone +1-555-123-4567

HomePhone**+** +1-555-123-0000

Result: HomePhone -> [+1-555-123-0000] [+1-555-123-4567]

HomePhone +1-555-123 -4567

HomePhone

Result: HomePhone -> [] (empty list)

Summary

It can be argued that the proposed certificate infrastructure has features in common with certain other products and systems. The central component of EMCSSL is client SSL-certificates which (while not very useful in their original design) become very useful and convenient under this proposal.

We can also note that the EMCSSL mechanism looks a lot like that used in authentication protocols such as **Kerberos**, where the client receives an "authorization token", and can then present this token to servers during a login process. But instead of using a centralized server, Emercoin's decentralized NVS is used, which acts as a unique token to successfully detect certificate serial numbers. Instead of verification of a client via a central server, the client can be verified by checking the NVS in a local instance of the Emercoin wallet.

The proposed InfoCard system also has much in common with OpenID or LDAP, but is decentralized and possesses an import mechanism those counterparts lack, allowing you to effectively maintain the consistency of large groups of cards.

When used together, EMCSSL and InfoCard deliver a safe and convenient system that allows users to login to sites and automatically fill their profile when creating new accounts.

How to get started

All software is distributed free of charge and the Emercoin wallet itself is opensource.

The package for generating client certificates, and InfoCard examples can be found at:

https://pool.emercoin.com/emcssl

You can check the operation of your certificate on the test page there, and also download the full example PHP implementation for yourself. Then based on the code contained in the example, you can add EMCSSL authorization to your own web services.

Visit the following forums if you have questions:

English https://bitcointalk.org/index.php?topic=362513.0.

Russian https://forum.bits.media/index.php?/topic/3408-emc-emercoin-pos.

For further advice please contact: team@emercoin.com