Proceedings of the 7th International Conference on Applied Informatics
Eger, Hungary, January 28–31, 2007. Vol. 1. pp. 55–63.

DESignIn asymmetric authentication system*

János Folláth, Andrea Huszti, Attila Pethő

Faculty of Informatics, University of Debrecen
e-mail: {follathj,husztia,pethoe}@inf.unideb.hu

Abstract

A simple but powerful authentication software is proposed. It uses USB drive as a hardware key, in combination with free software tools and asymmetric cryptography. With application of widespread and universal software and hardware components and strong cryptography, it forms an inexpensive and versatile high security authentication system. In this paper its security, protocol and implementation issues are considered.

1. Introduction

The DESignIn is a universal authentication and authorization software designed for the world wide web. It is purpose is to be an alternative to the common used password based authentication. Nowadays a user can have many accounts in association with different sites on the web, so he has to remember a dozens of cryptographically strong passwords. Most users have difficulties even with a single such password, so they write them down on a paper (or even worse in a file), or use weak passwords (using weak password makes the system vulnerable against dictionary attacks) or the same one for each site. An obvious solution is a good password manager. With this tool the password based authentication system loses one of his major virtues: the portability. With the passwords stored by a password manager on a single computer, the user cannot log in his accounts from another computer any more. In the password based systems the user himself is a great weakness. He means a weak point, that can be attacked by social engineering methods like phishing.

Using a DESignIn authentication system there is no need to remember passwords, and because of its asymmetric cryptographic primitives the key information never leaves the client machine, and so the phishing type attacks mean no danger. Another advantage of the DESignIn against the password based systems is, that it remains portable even if the user has many accounts over the web.

*All authors are supported by the "Aktion Österreich-Ungarn" project 63öu3.

DESignIn implemented to be userfriendly. Right after the SSL connection is set an authorized Java applet is automatically downloading that retrieves all the necessary information from the pendrive. After the successful authentication process, the client is able to achieve all the secure data through an SSL channel. The client does not have to do anything else, but inserting the USB stick.

The system is used by the members of the Faculty Council of the Faculty of Informatics of the University of Debrecen since March 2006 without any security problem.

2. Implementation

2.1. SSL hybrid

There is a widespread protocol playing a key role in the secure communication over the web. The Secure Socket Layer (SSL) is used to send sensitive data through untrusted channels (i.e. Internet), especially in combination with http. It was designed and implemented by Netscape. The current version (3.0) was released in 1996 [8]. The SSL protocol provides server (and optionally) client authentication during the session establishment and an encrypted communication during the session. It operates between the transport and the application layer, and it is intended to be a new layer in the model [5]. As such it supports not only the HTTP but any arbitrary application layer protocol. To the authentication of the server and the client a Public Key Infrastructure (PKI) is used. The SSL protocol has two parts: one for the authentication (SSL handshake) and another one for data encryption (and data integrity control). The SSL server authentication only protocol is often used to implement password based applications on the World Wide Web. As mentioned before the SSL protocol also provides a possibility of client authentication. In this case the client is also authenticated through PKI. This means that no password is used, the user must have a certificate validated by one of the servers trusted CAs. The storage of the user certificate is platform dependent (i.e. by Internet Explorer they are stored in the registry) [7]. In the password based (server authentication only) case the system is either weak or (when password manager is used) the portability is only virtual. As well as in the client authentication case: the system cannot be accessed from another computer due to the storage method of the user certificates. The goal of the DESignIn is to provide real portability and extend the SSL-s functions with authorization in a way that the authentication of the client is independent from the platform used. The possibility of several different clients' authentication is provided even through the same SSL channel.

2.2. Client implementation

The problem can be solved at three different level:

Transport Layer At this level one must modify the SSL protocol (or its successor the TLS (Transport Layer Security)) to suit the requirements above. This

solution has theoretical and also practical disadvantages:

- If we consider the TLS as the top of the transport layer, the services like authorization and the user certificate handling logically does not fit in the transport layer.
- We want to achieve real portability. When one modifies a current protocol, one must spread the new version before it takes real effect.

Application layer (Browser level) One can alter the browsers certificate handling method (this can be done via browser plugins). There are many web browsers, so to achieve a real portability one must to write a browser plugin for each browser (or at least for the most popular ones). A major drawback is that even if there are plugin for all browsers, it is no guarantee at all that on an arbitrary computer the plugin is installed in the browser (or the system administrator can be convinced to install it).

Application layer (Top level) The third option is to implement the new features with a mobile code on the client side. A drawback is that in this case we cannot rely on the SSL-s client authentication, because the services of that layer are hided by the browser. This means that a new mechanism is needed to authenticate the user, which of course is also an issue on the server side.

In the third case there is no major practical obstacle, so we decided to implement DESignIn at the top level on the client side. The implementation must have the capability to use system resources to access the user certificates, and must work on as many platforms as possible. There are more popular tools to run mobile code:

Java applet With Java Runtime Environment (JRE) installed in the browser Java applications can be ran, inside the browser. Usually these applications - for security reasons - run in a restricted environment (known as sandbox). Unfortunately a Java application running in a sandbox does not have the capabilities needed by this task, so one must use an RSA signed Java applet. If an applet signed with RSA, the signature proves the integrity and the origin of the Java code. With the certificate the identity of the vendor can be checked and if he is trusted by the user, then the code runs with full permissions (that the user has), otherwise it does not run at all.

ActiveX ActiveX is a binary code embedded in HTML pages. As the signed Java applet, it uses code signing to decide if a code can be trusted. Although it is primarily Windows/x86 platform specific, it can be get working on some other platforms, too. Anyway its platform specific nature means multiple work by implementation and inconvenience (or even worse: non-availability) by use.

JavaScript It is a scripting language that is implemented by the browser, and as such it is highly dependent on the browser used. JavaScript does not have

an exact security policy: it differs from vendor to vendor. It uses either the sandbox model or the code signing method. Of course when the sandbox model is used, the script does not have the capabilities to do the job.[1]

One can accept the disadvantages of either tool or detect the client platform and use the most appropriate one. In all case one must use a code signing method (otherwise can only run in the sandbox). It is important, that the signature on the code identifies the program and not the server (the server authentication takes place at the transport layer). It is a distinct level of trust, to download one's content (server authentication) or let run a potentially malicious foreign code on the computer (code sign). By the client authentication even if the server could not be authenticated but was trusted (i.e. self signed pages), the code signature should be always authenticated by a trusted third party who takes the responsibility for the code (PKI). By the implementation of the DESignIn the first option was chosen, because the Java applet has the best portability properties.

2.3. Hardware key

A single hardware key can hold many private keys and user certificates, and is easy to carry. It is not only practical, but it is a physical manifestation of the private key, the users identity. It is much easier to make a normal user understand the importance of a key-like USB stick (or a smart card that is similar to bank cards), than some password or mysterious file on the computer. DESignIn uses a public key cryptography and tends to be a portable authentication system, so the user's private key must be stored on a portable device. The keying material is usually stored on smart cards. Unfortunately a card reader is not a standard equipment on most computers, so its usage would harm the mobility of the system. A common storage media is needed, that is easy to handle, is available on most computers. The DESignIn uses USB sticks to store the user's private key information. A simple USB drive can hold many keys, do not need extra driver, and can be easily copied like the real keys. The ability to read and write the key may seem insecure but by an average user it is necessary: one can handle his own private key, and can use his USB-drive for other purposes too.

2.4. Remaining issues

The Java applet runs inside the JRI, which is a virtual machine. Its portability is granted by the fact that the JRI can be implemented on many platforms. And thats why the JRI-s interface to the operating system must be universal. Universality in this case means simplicity, so the applets device handling ability is very restricted. It cannot be determined with pure Java code, whether a USB is plugged in, and if it is, then where it is in the filesystem. In the current version the DESignIn simply tries to read the key and the id from the possible USB locations. On UNIX-like

[1] For example the Netscape versions before the Netscape 4 and the Internet Explorer does not support code signed JavaScripts

platforms there is a place where the USB drives used to be mounted, and these places are searched. On Windows platforms this means to search all the filesystem roots. And this procedure is where the problems come from. On UNIX-like systems there is no guarantee that in all instances of that system the USB will be mounted to the documented location. In this case if the private key is not available a simple Java exception occurs. In the other case: if the applet tries to read an unavailable device, the program does not notice anything, and the user will be prompted to insert a disc. As mentioned earlier, there is no pure Java way to solve the problem. Fortunately one can embed native code in Java, and the problem can be solved by writing a USB checker function for each platform.

2.5. Server implementation

DESignIn server is implemented using PHP server-side scripting language and PostgreSQL relational database system. In order to run cryptographic functions the web server should be set up with SSL module. After setting up the SSL channel, the concatenation of a random number and the actual time is generated, saved in a database and the applet is downloading from the server, if the client accepts its certificate. Following the server receives the RSA digital signature on the nonce from the client. The authentication protocol does not insist on using RSA, any other signature mechanism might be applied. After verification of the signature the server according to the ID transfers to the proper web site. Nonces are stored in a database, in case the authentication process is not successful in a fixed time, they should be deleted according to their time stamp.

Since a session is started the client is given a session identifier. The browser stores a single session id that finds and initializes the variables stored on the server. Sessions have a timeout otherwise, if a user leaves the web site, there is no way the server can tell when the session should end. Thus after certain amount of time sessions are destroyed, the client should log in again to achieve the documents.

After authentication all the secure documents are transferred encrypted by the SSL secure key. The protected documents stored on the server in a selected folder. The exact place of the folder is kept unknown. After the authorization a new, temporary folder is created with a name generated from the client's session id. The links of only those documents are copied into this folder that the client has required. Right after logout, this folder including the links are deleted.

The proposed authentication system might be implemented in other server-side programming language or database system, too. All the components necessary for operation are chosen, because they are free and Open Source.

3. The protocol for authentication

3.1. Description of the protocol

The aim of the protocol is to authenticate a user, achieving it by interacting with another entity called server.

Participants. The protocol involves two entities, one of them a user(client), who wishes to prove his identity, the other one is a trusted server. The server is trusted in a meaning of executing the protocol faithfully, and not to be engaged in any other activity that will deliberately compromise the security. Furthermore, the server is trusted to generate a cryptographically secure pseudo-random nonce(number used only once).

Channels. Participants use public channels, in order to achieve secrecy all data sent through is encrypted by a symmetric key commonly agreed on during the SSL handshake.

Unilateral authentication protocol. The authentication protocol consists of two main parts, the SSL Handshake Protocol [8] and a protocol based on ISO/IEC 9798-3 [4]. The SSL Handshake Protocol is specialized to allow clients and servers to communicate without suffering from impersonation, eavesdropping, tampering, or message forgery. International Standard ISO/IEC 9798-3 uses digital signatures to determine an entity's authenticity. One entity can prove its identity to another entity by using a private key to generate a digital signature on a random challenge.

Handshake protocol overview. When a SSL client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, server authenticates itself, and use public-key encryption techniques to generate shared secrets.

First the client and a server in the client hello/server hello messages establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, client/server random values are generated and exchanged. Then the server will send its certificate.

The client then verifies the server's public key, and responds with a client key exchange message, the content of the message will depend on the public key algorithm selected between the client hello and the server hello. At this point, a premaster-secret is sent by the client, the client generates master-secret from the premaster-secret and immediately sends the finished message under the new algorithms, keys, and secrets. The server decrypts the premaster-secret and generates master-secret. In response, the server will send its finished message under the new algorithms, keys, and secrets. At this point, the handshake is complete.

Protocol based on ISO/IEC 9798-3. The second part of the unilateral authentication protocol is a modified version of the two-pass authentication ISO/IEC 9798-3. The steps of the protocol are as follows:

1. The server generates a nonce as a concatenation of a random number chal-

lenge (n) and a timestamp (t), stores it in a database and sends it to the client encrypted by the SSL secret key:

$$S \longrightarrow C : Enc_{SK}(n||t)$$

2. The client upon receiving the message creates an authentication token, including his identification number (ID) and a digital signature. Both the ID and the secret key used for signing are stored on his USB stick. A Java applet, authorized by the server, is downloaded, accesses USB stick inserted in the USB port calculates the signature and sends it with the ID encrypted to the server.
$$C \longrightarrow S : Enc_{SK}(n||t, ID, Sig_C(n||t))$$

3. The server looks up the corresponding public key with the help of the ID in the database, verifies the signature. The server checks whether the nonce exists in the database and whether it is valid. A nonce is valid if it is sent no more then a certain amount of time ago. If any of the verifications fail, then the authentication exchange is terminated, otherwise the server deletes the nonce from the database. Successful completion of the steps means that the client has authenticated himself to the server.

Following the authentication process both parties securely communicate to each other through an SSL channel.

3.2. Security issues

D. Wagner and B. Schneier have already examined the security of SSL in [6]. A list of security properties and attacks are listed in [1]. Considering the whole protocol, it meats the following are security requirements.

Confidentiality ensures that data is only available to those authorized to obtain it. It protects against eavesdropping attack. This is achieved through encryption of the data by the SSL secure key.

Data integrity ensures that data has not been altered by unauthorized entities. SSL protects data integrity by using cryptographic MAC.

Data origin authentication guarantees the origin of data. The origin of data is guaranteed by SSL MAC.

Non-repudiation ensures that entity cannot deny sending data that they have committed to. Property of non-repudiation is important when the client sends his token. It is provided using digital signature mechanism.

Freshness ensures that the message sent is not replayed from an old session. Freshness value is guaranteed by a random number and the timestamp. For each authentication a new random number is generated and a timestamp is concatenated to it. No one can prove his identity with the same random number again, since it is deleted form the server's database. The timestamp is an extra security

feature. After certain amount of time if there is no proper response, the nonce is automatically deleted from the database.

Replay attacks. The adversary records information seen in the protocol and then sends it to the same, or a different, principal, possibly during a later protocol run. SSL is the first security channel, so all the information goes through is encrypted, an adversary cannot access any of the data. Even if an adversary sends any part of the message again in the same SSL channel, with an already used nonce it is impossible to complete the authentication, since after the process they are deleted from the database.

Denial of service attack. Denial of service attacks in practise take place against servers who are required to interact with many clients. This attack aims to exhaust the number of allowed connections to the server. Since the timestamp part changes regularly with a sufficiently long random number, sufficiently large amount of nonce can be generated. This means that a slow or low-cycle pseudorandom generator makes the system vulnerable against this type of attack. Since the output of the generator is always encrypted, fast large cycle generators with good statistical properties that are weak against algorithmic attacks (like Herendi [2]), are a good choice here.

Certification Manipulation attack. In public key protocols the certificate of a principal acts as an *off-line* assurance from a trusted authority that the principal's public key really does belong to that principal. This leads to potential attacks in which the adversary gains a certificate that a public key is its own, even though it does not know the corresponding private key. In our case certification is used for server authorization on the level of SSL. During the SSL handshake the server should decrypt the premaster-secret with its secret key, it means without possessing the private key the SSL connection cannot be set.

Protocol Interaction attack. Most long-term keys are intended to be used for a single protocol, however, it could be the case that keys are used in multiple protocols. Protocols designed independently may interact badly. A USB stick is capable of storing several private keys and also certificates. DESignIn is designed to be an independent protocol, the corresponding private key material is not allowed to be used for other purposes.

3.3. Further development

An improvement to the current version is to encrypt the private key with a blockcypher. This is an analogous solution to the password managers, just this keystore is mobile. It also means to add a passphrase to the authentication procedure. This extension is already under development. The security can further upgraded by changing the transporting media: when equipped with a tiny processor, the USB can perform the computations onboard, so the key never leaves the secure environment. There are already USB devices that can perform cryptographic computations (like iKey) but they need a special driver to be installed on client side, and it is totally unacceptable by DESignIn.

References

[1] Boyd, C., Mathuria, A., Protocols for Authentication and Key Establishment, *Springer*, (2003).

[2] Herendi, T., Uniform distribution of linear recurring sequences modulo prime powers, *Finite Fields Appl.*, 10, no. 1, (2004), 1–23.

[3] Johnson, B. C., Gossels, J. G., Davis, D. T., The SSL Handshake, *Perspective on practical security*, (2004).

[4] ISO, Information Technology - Security Techniques - Entity Authentication Mechanisms - Part 3: Entity Authentication Using a Public Key Algorithm ISO/IEC 9798-3, 2nd edition, (1998). International Standard.

[5] Tanenbaum, A. S., Computer Networks, *Prentice Hall*, (2003).

[6] Wagner, D., Schneier, B., Analysis of the SSL 3.0 Protocol, in: *Proceedings of the 2nd USENIX Workshop on Electronic Commerce*, (1996).

[7] `http://www-128.ibm.com/developerworks/lotus/library/ls-SSL\_client\_authentication/`

[8] `http://wp.netscape.com/eng/ssl3/`

János Folláth, Andrea Huszti, Attila Pethő
Faculty of Informatics, University of Debrecen
P.O. Box 12, H-4010 Debrecen
Hungary
Hungarian Academy of Sciences and University of Debrecen
Hungary