DEVELOPMENT OF APPLICATION FOR REMOTE DATABASE ACCESS USING JAVA SECURITY FUNCTIONS

Jasmin Malkić, Tatjana Welzer, Boštjan Brumen

Abstract

Business use of Internet represented by on-line banking, shopping and other commercial activities, introduced a few years ago, has made the development of a solid and secure data transfer through the public network largely necessary. The resu lt of the development made in this direction, was a wide spectrum of solutions that provide different levels of security. By rule, all those projects that could tackle high security standards that e-commerce requires, applied cryptographic engines. The paper analyses the history of Internet security in brief, and pays a special attention to the cryptographic properties of the programming language Java, now and in the future. By setting an example of the client-server application for remote authorized access to a database source, it demonstrates the use of Java's basic cryptographic tools.

Izvleček

Poslovna uporaba interneta v spletnem bančništvu, trgovini in drugih poslovnih dejavnostih, ki je bila vpeljana v zadnjih nekaj letih, zahteva razvoj varnega prenosa podatkov preko javnega omrežja. Rezultat programskega razvoja v tej smeri je širok spekter rešitev, ki ponujajo različne stopnje varnosti. Kriptografske algoritme pravilo-ma uporabljajo projekti, ki izpolnjujejo visoke varnostne standarde za varen prenos podatkov. Članek kratko analizira zgodovino varnosti na internetu, s poudarkom na kriptografskih lastnostih programskega jezika java, sedaj in v prihodnosti. Podan je tudi primer aplikacije odjemalec-strežnik za avtorizirani dostop do oddaljenega podatkovnega vira, ki prikazuje uporabo osnovnih javanskih kriptografskih orodij.

■■■

0. Introduction

Since its very beginning, Internet has been dealing with two, at first sight totally opposite tasks - providing public information and hiding its secret content from unauthorized access. Although data transfer for both, public and secret content can use the same lower communication layers, it's obvious that the transfer of secret information requires some extra work in order to keep it secret.

There are many different possible scenarios for data transfer over the Internet when we need to protect data, and they require the implementation of different security levels. Although the universal security solution isn't given, good solutions for different situations do exist. It's hard to unify the Internet security approaches, mainly because of the very nature of Internet itself. Its basic philosophy - to be open to many computers which work on different operating systems and use different Internet clients, implicates a lot of different security approaches. What is also important is the fact that Internet operates in many countries, and security laws can vary. This leads to the conclusion that every Internet security project should be made in accordance with the given local conditions.

1. The Origins of Internet Security

Before HTTP (Hypertext Transfer Protocol, defined in RFC 2616) became the number one format for Internet data, the public need for security didn't really exist. Internet was mainly the occupation and privilege of the academic and military structures; security attacks were possible but rare and much easier to track. Upon emergence of the World Wide Web, which uses HTTP data to present its content, Internet quickly entered the everyday life of many people who could now use it to read the latest news, for personal presentation, and even to do their shopping or bank transactions. It became clear that protecting private data that begun to circulate through the wires was a must.

Web servers and HTTP were not designed to tackle serious security tasks. Although basic authentication is supported by many Web servers, (e.g. *Apache Web Server on UNIX*) where access to a certain directory can be secured by a password stored on the server, the problem arises because nothing protects the password itself while it travels through the Internet in HTTP authentication request. Anybody who cares enough can intercept such a request and read the plain text password in it.

As soon as HTTP became the Internet mainstream, several groups started to work on the methods for securing its transfers. The official project, sponsored by the IETF was named Secure HTTP (*Farrow, 2001*), but also some other works found their way to users. One of those created by Netscape, thanks to the large number of Netscape Navigator users, managed to exceed its initial purpose – to improve Navigator's security. Today, this cryptographic security solution named SSL has its free implementation (OpenSSL) for multiple operating systems, and a large developer community, but still it is not the Internet standard. Which is partly because some serious problems arise with the use of this security method.

The first factor that proved that SSL was far from being perfect, was growing hardware ability to brute force the cryptographic keys. Brute forcing method guesses all the possible combinations of the key, and with 40-bit keys that were used by most Netscape and MS Internet Explorer clients, can be guessed within less than one day with the latest processors involved (*Farrow, 2001*). To avoid this, the use of most recent operational systems and Internet browsers which introduce 128-bit encryption keys is largely recommended. Still, SSL has also some serious problems connected with the fact that its reliability highly depends on the environment, first of all, the client browser being used. Such problems usually result in CERT Advisories or security patches for commonly used Web browsers.

In May 2000, *Kevin Fu* from MIT discovered that Netscape Navigator could be fooled to accept an invalid certificate, or in other words, the right certificate from a false location (*Farrow, 2001*). If a user visited a site that had a certificate where the server name didn't match the common name found within the certificate (usually the case of using the stolen certificate), Navigator failed to detect this and let the user pass secret data to suspicious server. *Mitja Kolšek* from ACROS, Slovenia, discovered another problem with Navigator. Kolšek has discovered that Navigator assumed that subsequent access to the same IP address was part of the same SSL session. Combined with possible DNS spoofing attack between subsequent sessions, this could lead into Navigator visiting a different site (due to failure in checking if server common name matches the one within the certificate), and leaving data on a wrong server. CERT Advisory CA-2000-5 covered this problem. This showed that in spite of the existing certificate verifying system that SSL provides, client's security holes can produce some serious doubt on behalf of a user, to whom a secret content is transferred.

Microsoft's Internet Explorer has also had problems with the implementation of SSL, in versions 4 and 5 the password, part of the basic HTTPS authentication scheme, once sent to server was revealed (sent without encryption) later if a client made a HTTP connection (without SSL) to the same server.

This has been solved by issuing the Microsoft's security advisory and the official patch addressing this problem. Those are only some of the problems that arise from SSL employing the common Internet clients (Netscape Navigator or Internet Explorer) as an important part of a secure connection. However, this appears to be the price SSL pays for its portability, and many clients and operating systems supporting it.

2. The Use of Security Hash Algorithm (SHA-1)

As a US Federal Standard in the category of Computer Security, SHA-1 was established by National Institute of Standards and Technology in April 1995 (*Burrows,1995*). Since this body encourages all private and commercial organizations to use it, SHA is widely used and implemented, whenever any kind of software security is needed, also in many tools other then Java or SSL described here. Besides Internet, SHA-1 also finds its place in data storage, software distribution, and other applications that require data integrity assurance and data origin authentication.

Input for SHA-1 is a binary message with less than 2^{64} bits, out of which algorithm computations produce a 160-bit output called *message digest*. SHA-1 is an improved technical revision of SHA, as described in *Federal Information Processing Standards Publication (FIPS) 180*. SHA-1 is secure due to its properties: it is computationally infeasible to find a message, which corresponds to a given message digest, or to find two different messages which produce the same message digest.

SHA-1 message digest can also be used as an input to the *Digital Signature Algorithm* (DSA) which generates or verifies the signature for the message. If the message integrity is more important than its secrecy, it pays off to digitally sign the message digest rather than the message itself, because the message digest is in most cases much shorter.

SHA-1 "engine" is based on the principles similar to those used by *Prof. Ronald L. Rivest* from MIT when designing the MD4 message digest algorithm (*Springer-Verlag, 1991*). It sequentially accepts packets of 512 bits, which means that a start message has to be padded to contain 512 x n bits. In order to achieve this, a "1" followed by the m "0"s are appended to the end of the original message. At the very end of a padded message, a 64-bit integer which expresses the length of the original message (in bits) is also appended, which concludes padding process (*Burrows,1995*).

3. Java Security Features

The security considerations in Java Cryptography Architecture mainly point at secure authentication and data transfer (*Sun, 2001*). As Java is, after all, an object language designed preferably for network and Internet programming, it's therefore essential to provide data integrity on the way from one machine to another. In order to achieve this, there are three main cryptographic concepts which Java uses: *Message Digests*, *Digital Signatures* and *Certificates*. They assume different levels of security, and their use depends on the security objectives.

Message Digest

The basic security tool, not only in Java, is the *message digest*, and almost all Java security concepts use it in some form. When a situation allows, pure digests of the secret content can be instantiated and used for authentication purposes. The class which is responsible for this is *java.security.MessageDigest*, where all the necessary methods for this process are contained. Cryptography core is provided by Java implementation of ***SHA-1*** or ***MD5*** algorithm, between which a user chooses when creating an instance of MessageDigest class/object:

```
MessageDigest md= MessageDigest.getInstance(String algorithm);
```

The *MessageDigest* object has the ability to be filled with the array of bytes using the method *update(**byte**[ ] data_to_diggest)*, and to digest its content using the method *digest( )*, with no parameters. If chosen algorithm for *MessageDigest* object is SHA-1, loaded array of bytes will be the subject of cryptographic procedure producing a 160-bit output from the *digest( )* method. This can make a good use for the secure transfer of passwords.

If a remote client needs a password to be authenticated to the server, it can digest it for secure transfer through non-trusted network, then transfer it to server which from its side can also digest its own copy of password and compare it with those received from the client for authentication.

Some downs of the message digesting are decreasing the public confidence in MD5, and the possibility of a third-party catching the digested password during the transfer and interpret it later to the server in order to access its secured resources. To avoid this, the improvements can be made by computing the digest together with timestamp and random numbers.

This approach focuses on a situation when the authentication of a client to the server is of greatest importance, and server's identity is unquestioned. As it's also the most obvious way for the demonstration of Java's implementation of SHA-1 algorithm, it has been chosen as an example in the next chapter.

Digital Signatures

A signature is the message digest encrypted with the signer's private key. In Java, signatures are provided by methods defined in *java.security.Signature* class. Factory methods for *Signature* object are *getInstance* with two different constructors:

```
Signature p = Signature.getInstance (String algorithm); and
Signature p = Signature.getInstance (String algorithm, String provider);
```

The second method returns a *Signature* for the given algorithm-provider pair, while the first does the same but with the first provider that supports the given algorithm. A signature uses two algorithms – one to calculate a message digest and one to encrypt the message digest with the issuer public key. The SUN provider shipped with JDK1.1 and further versions supports ***DSA*** encryption of a ***SHA-1*** message digest (*Oaks, 1998*). This is simply referred to by giving a DSA string value to the first *Signature* constructor. Data to *Signature* object are fed on the same way as to *MessageDigest* object, using the *update()* method, however Signature first has to be initiated with the signer's private key by method *initSign(**PrivateKey** myPrivateKey)*. On the server side, received *Signature* object is verified by *initVerify(**PublicKey** myPyblicKey)* method, which returns *boolean* variable to indicate if signature is verified or not.

In the *Signature* class, Java introduces private and public key for authentication, where the private key of a signer is used for making the signature, and public for verifying it on remote side. Signatures still don't provide full confidentiality because of the public key transfer. If a client would always carry a signer's public key e.g. on a floppy disk, and could use it when necessary this method would be much more secure. On the other hand this would be a bit unpractical for use on the Internet. Certificates do solve this problem (*Kundsen, 1998*).

Certificates

To verify a signature, the public key file is needed, but the problem is how to distribute the keys securely over the network. Even if the key can be downloaded, how can a user be sure that the received key is really issued from the server that it refers to. The best solution to these problems, which still maintain all the benefits of Internet, is given in the form of the Certificates.

A certificate is a statement, signed by one person, that the public key of another person has a particular

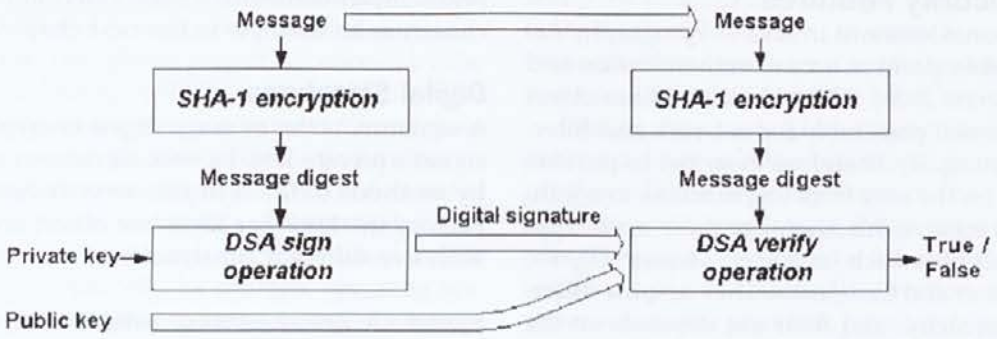

Figure 3.1. **–Common Digital Signature process with use of SHA-1 and DSA**

value. A person who guarantees the validity of the other certificates has to be trustworthy, and is called a *Certificate Authority (CA)*.

Java API from version 1.2. is equipped with necessary methods to recognize the certificates issued from CAs that use ***X.509v3*** certificates. Most of the methods for distributing the certificates are packed into *java.securtiy.cert.Certificate* class.

The certificate object contains information about the issuing identity, its public key, and its signature of the above information. All that information can be recalled using the methods defined in that class, such as *getPublicKey( )*. Support for ***X.509*** certificates is stored in the additional class. With *getInstance( )* method in *java.security.cert.X509Certificate* class, such certificate can be loaded into Java program. The validation on the server side, is implemented through the standard method *validate( )*.

By using the certificates, an additional trusted authority is provided to assure the purity of signature. This feature is expected to develop even more in future versions of Java (JSDK 1.4.1 and further), in order to assure full support for this way of securing the communications over public network.

4. SHA-1 Encryption Within Java – An Example

A simple client-server architecture, designed for the remote access to database source could be an example of Java security basics. Regarding the building of such architecture itself, Java offers two generic classes - *Applet* for client and *HttpServlet* for server side.

Java applets depend on Java Virtual Machine (JVM, shipped with almost every Internet browser available) that has to be implemented on the client side. JVM's runtime security properties let the downloaded *Applet* classes run inside it, while preventing any harm to the local system (*Oaks, 1998*).

The server side needs a web server to serve the applet class to remote clients, and on the same machine an environment for the central application should reside. Such application has to be able to manipulate with the HTTP requests - responses, and to communicate with the database. While every Java application which imports *java.sql* package and proper JDBC database drivers can do database communication, HTTP handling is reserved for the servlets – Java's server-side components in many ways analogous to CGI applications (*Fields, 1999; Darby, 1998*).

Figure 4.1. illustrates the password verification authentication concept. The implementation of the client and server Java classes will be explained here in brief, while the full source code can be seen on http://www.inet.ba/malkic/ird1.

Client Side

The important GUI objects in client class (*class CappletTs extends Applet*) are: *messageField* and *passwordField* (of type *TextField*) that contain the parameter for database query and authorization password respectively; and *responseField* (of type *TextArea*) that receives a response from the server – selected database content or error message. When the GUI event occurs (password and database query sent) method *private void interactWithServlet ( )* is called. To establish a HTTP connection with Java servlet application that resides on the server, an URL object must be created within this method. *servletURL* of type *URL* and *servletConnection* of type *URLConnection* encapsulate the server's address and communication port, while *servletConnection* also handles two data streams for output and input (*out* of type *DataOutputStream* and *in* of type *InputStream*).

The cryptography object in this applet is *md* of type *MessageDigest*. Its constructor receives "SHA" string parameter which means this object will encode its content using SHA-1 algorithm. For better security of the

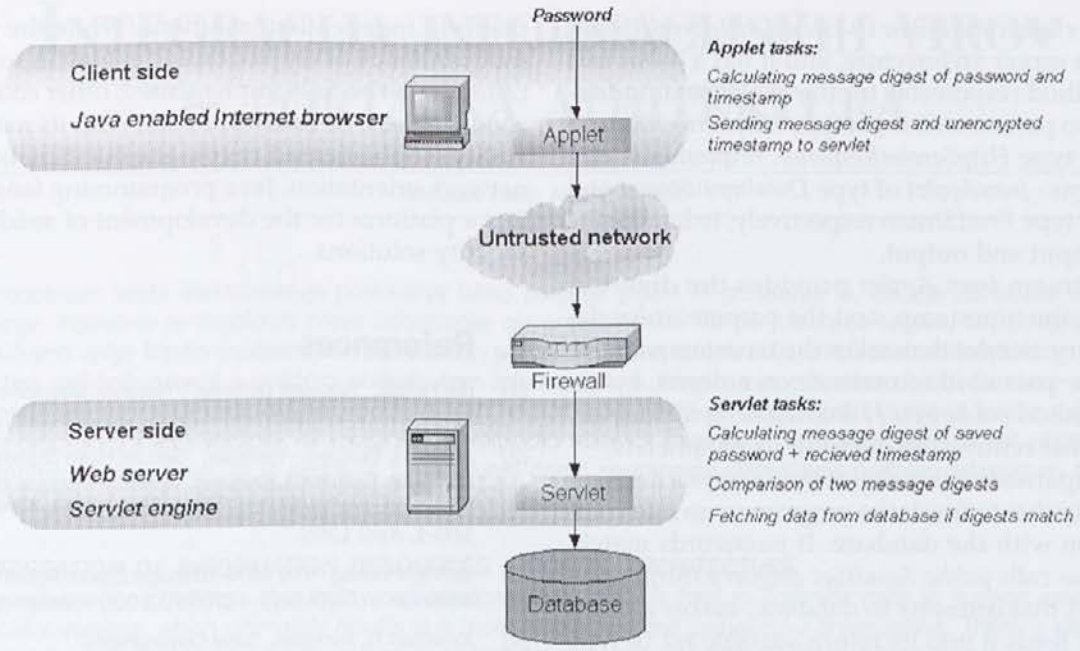

Figure 4.1. – **Simplified architecture of authorized access to remote database source**

transferred data, the password represented by *String passwd* is encrypted together with *timeStamp* of type *long*. This variable is a numerical interpretation of the current value of the local *Date* object. *md*'s *digest()* method output is *byte* array *protect* that is sent to server through *out* together with unencrypted *timeStamp*. Enclosed in *out* are also parameter for database query (from *messageField*), and *int* value *length* which indicates number of *protect* array members. When SHA-1 is used *length* isn't so crucial because it's by SHA-1 definition always 20 (length of SHA-1 output is 160 bits or 20 bytes), but this leaves space for the use of other algorithms too.

Method *public byte[ ] toBytes (long lval)*, has an internal purpose, and serves as a converter from type *long* to *byte[ ]*. *MessageDigest* object can't be updated with *long* type directly, so in this case *timeStamp* variable of type *long*, has to be converted first (server side uses the same method for this conversion). *Figure 4.2.* shows both encrypted and non-encrypted data that applet is sending through the public network.

Server Side

Web server installed on the server machine serves the applet class to remote clients. Together with Java servlet runtime components it also enables working conditions for the servlet class that resides on the server. A suitable combination of these components could be e.g. SUN's JSDK 2.1 with Microsoft's IIS. An important segment of the server is the database – it can be an ODBC data source, but also any other database that has JDBC drivers implemented (besides ODBC standard JDK contains the drivers for Oracle database, and a free driver classes package for MySQL is also available on *http://mysql.sourceforge.net*).

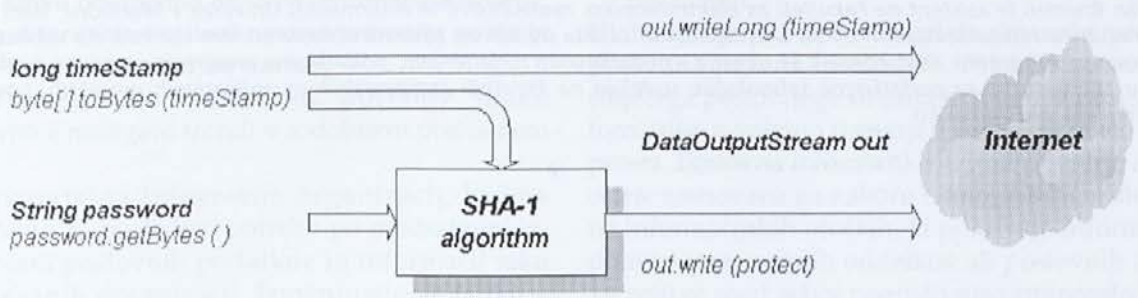

Figure 4.2. – **Secure password transfer from the client side**

Servlet class (*class DBaseTh extends HttpServlet*) concludes this server architecture, and it has a *public void doPost* method responsible for the HTTP communication. Its two parameters, *req* of type *HttpServletRequest*, and *res* of type *HttpServletResponse*, implements two data streams - *fromApplet* of type *DataInputStream* and *toApplet* of type *PrintStream* respectively, to handle the servlet's input and output.

Data stream *from Applet* provides the digest of password, the timestamp, and the parameter for database query. Servlet then takes the timestamp and its copy of the password to make its own digest, by using the method *public byte [] doSHAEncrypt(String sec, long lval)* that returns the digest of its parameters.

By comparison of two digest byte arrays a decision is made whether to finish the program or go into communication with the database. If passwords match, servlet class calls *public ResultSet doQuery (String query)* method that connects to database, makes a given query and feeds it into its return variable *rs1* of type *ResultSet*. The data from *rs1* are extracted by using its method *next( )*, and sent through *out* back to the client.

5. Conclusion

The need for the Internet security in the world of e-commerce grows together with the Internet itself. As in the Internet terms growth also means a variety of software involved, both on client and server side, it seems that secure data transfer is a good basis for the platform independent solutions. While the platform independency of cryptographic engine in such applications must be fully implemented, other components should at least be easily portable. With its native portability, sophisticated cryptographic functions, and network orientation, Java programming language offers a platform for the development of solid Internet security solutions.

6. References

[1] Rik Farrow, *"Network Defense"*, Network Magazine Vol.16, January 2001, http://www.networkmagazine.com.

[2] James H. Burrows - redactor, *"Secure Hash Standard"*, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg USA, FIPS PUB 180-1, April 1995.

[3] Springer-Verlag, *"The MD4 Message Digest Algorithm"*, Advances in Cryptology – CRYPTO 1990 Proceedings, 1991.

[4] Jonathan B. Kundsen, *"Java Cryptography"*, O'Reilly Books May 1998.

[5] Scott Oaks, *"Java Security"*, O'Reilly Books May 1998.

[6] Chad Darby, *"Applet and Servlet Communication"*, Java Developer's Journal, September 1998.

[7] Sun Microsystems, *"Secure Computing With Java: Now and the Future"*, SUN Microsystems White Papers 2001, http://java.sun.com/marketing/collateral/security.htm.

[8] Dyane K. Fields, "Applet-to-Servlet Communication for Enterprise Applications", 1999, http://developer.netscape.com/viewsource.

◆

Jasmin Malkić je diplomiral na Fakulteti za elektrotehniko in računalništvo Univerze v Zagrebu (Hrvaška) in je študent podiplomskega študija računalništva in informatike na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Sodeloval je pri projektih v javi in C++ pri razvoju "GSM Billing and Customer Care" sistema v podjetuju ZIRA Ltd., Sarajevo (Bosna in Hercegovina). Zadnje dve leti se ukvarja z varnostjo na internetu.

◆

Dr. Tatjana Welzer je izredna profesorica na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Predava predmete Podatkovne baze I in II ter Dostopnost in zaščita podatkov, na univerzitetnem in na visokošolskem strokovnem programu, ter predmet Zaščita v raunalniških okoljih na podiplomskem programu. Raziskovalno se ukvarja z načrtovanjem podatkovnih baz, ponovno uporabo, kakovostjo podatkov in varnostjo računalniških sistemov. Kot vodja Laboratorija za podatkovne tehnologije vodi ali sodeluje pri mnogih domačih in mednarodnih projektih, povezanih s problematiko podatkovnih tehnologij.

◆

Boštjan Brumen je asistent na Fakulteti za elektrotehniko, raunalništvo in informatiko Univerze v Mariboru. Vodi vaje pri predmetih Podatkovne baze I in II ter Dostopnost in zaščita podatkov, tako na univerzitetnem kot tudi na visokošolskem strokovnem programu. Raziskovalno se ukvarja s podatkovnim rudarjenjem, podatkovno analizo in varnostjo podatkov. V okviru Laboratorija za podatkovne tehnologije sodeluje na številnih raziskovalnih in aplikativnih projektih, povezani s podatkovno problematiko.

◆