Journal of Mobile, Embedded and Distributed Systems, vol. V, no. 1, 2013
ISSN 2067 – 4074

Mobile Solution for Digital Signature

Cristian UDREA
IT&C Security Master
Department of Economic Informatics and Cybernetics
The Bucharest University of Economic Studies
ROMANIA
cristian.udrea88@yahoo.com

Abstract: This article describes a mobile interpretation of the PKI architecture used in document digital signature. The architecture relies on a three-way client-server model which communicates via a combination of Bluetooth and Wireless LAN channels. The idea of implementing such a system using an Android device as the signee is an alternative to the more traditional approach which uses desktop applications and USB tokens for digital certificate storage and use.

Key-Words: Digital Signature, Bluetooth, Android, Public Key Infrastructure, Mobile Architecture.

1. Introduction

Today's society is, doubt aside, heavily reliant on technology and computer science. Few aspects of our everyday lives still function in a more traditional sense (we might think of art). In such times, there are limitless possibilities to participate in the ongoing development process of technology. This phenomenon is one of global magnitude, bringing people together around the world, to share views and, most importantly, ideas.

The current paper describes a theoretical and practical approach of the "digital signature" concept using a mobile architecture.

Digital signature was first used somewhere around the 1970's, corresponding to the development of the RSA encryption algorithm. The first to implement and release software that provided document digital signing was Lotus Notes 1.0, in 1989. From a definition point of view, a digital signature can be thought of as data in digital format, attached or logically associated to some other data that provide a means of identification. The signature has to uniquely relate to the signee and provide means of his identification, must be created by means under the sole control of the signee and has to be linked to the signed data in such a way that all further modifications brought to the document after it has been signed are easily identified.

Looking at digital and traditional signature, we can easily conclude that the practical aspects of the two are identical. Furthermore, the latter is highly recommended due to its multiple security endorsements.

2. Digital signatures on a mobile architecture

2.1 Why we need digital signatures

Digital signature comes with numerous advantages when we compare it to traditional signature. First of all, the chances to replicate a digital signature depend only on how well the signature and encryption algorithms are implemented. A thorough design and implementation reduces the chances of forging a digital signature close to zero. Moreover, in order to digitally sign a document, the signee does not necessarily have to be present. The same applies to sending the document to the client (we can use email, web services, Bluetooth and any other form of digital data communication).

There are further advantages that come with using digital signatures. Consider the costs of paper and ink. It might not seem

like much, but if we take this small amount and multiply it by the number of documents that are signed everyday worldwide, the resulting figure might not be so easy to overlook. Also, it is much easier to keep an archive of digital documents rather than traditional documents.

2.2 Existing digital signature solutions

Regardless of our choice in implementation, we must keep in mind a series of objectives when we want to commit to digital signature architectures. Used algorithms must provide a high degree of safety. Some public-key based algorithms have known vulnerabilities which can easily be exploited. After choosing a reliable algorithm, we must ensure that we implement it according to its stated requirements. A weak implementation of a strong algorithm will, at best, lead to mediocre results. Afterwards, we must ensure that the private key remains private. If by accident the private key confidentiality is compromised, all digital signatures may be forged and should be declared invalid by default. Furthermore, the public key must be verifiable. The term of certificate authority needs mentioning in regard to this matter and will be discussed later on. Last but not least, the users of the digital signature architecture must respect the protocol standards as-are.

Considering the possible implementation alternatives, digital signature can be accomplished in two major ways: those that use the PKI format (Public Key Infrastructure) and those that do not. Non-PKI architectures are considered to be weaker in terms of security and non-compliant with legal requirements. This happens for a number of reasons, such as not being unique for one user, not being able to identify the signee, not being able to detect changes brought to the signed document and not guaranteeing that the signee is the sole owner of the signing certificate (private key).
Most digital signature providers offer several features. Timestamping helps avoid possible deniability of a document's digital signature. It confirms the existence of signed data at a given moment in time. Additionally, the timestamp can include an ID provided by the timestamping authority. Online certificate validation is the responsibility of the client and is achieved by interrogating Certificate Revocation Lists using OCSP (Online Certificate Status Protocol). EmailerSafe is a feature that guarantees that an email containing the signed file was sent and received, thus ensuring data integrity, authenticity, confidentiality and nondeniability. Digital signature can also be adapted for use in electronic billing.

2.3 The Public Key Infrastructure

PKI guarantees the association between the existing private keys and the owners by including a third party – the certificate authority (further referred to as the CA).

There are three main components in a PKI infrastructure, as follows:

- **The registration authority (RA).** The RA is the authentication process by which the user requests for obtaining a digital certificate are handled. The RA sends a certificate issue request to the CA.
- **The certificate authority (CA).** The CA is the component that issues the digital certificate. This certificate includes a public key and the certificate user's identity. The certificate is the one component that is used later on to verify the public key used in signing documents.
- **Database support.** A database is also needed to keep track of all issued certificates and the system users.

2.4 PKI variations

Companies can opt for more than one PKI alternative, each bringing a series of advantages and disadvantages.

Administered PKI – based on outsourcing

Outsourcing refers to using a PKI solution which is the property of and under the

ISSN 2067 – 4074

administration of a trusted third party, also known as the certificate authority. The CA is responsible for enforcing the functional policies, the administration rules, the set of used technologies and the legal aspects regarding its clients. This approach does not require the acquisition of any software or hardware resources. On the other hand, long term, the working costs can be quite high if we think about the initial costs and the yearly license renewal fees, as well as the cost of maintenance services.

Traditional PKI – developing your own solution

Developing your own PKI architecture involves the acquisition of both hardware equipment and software licenses. A team is required for creating, managing and providing maintenance for the system and its users. This approach comes with a higher degree of flexibility in terms of signature customization to satisfy the internal infrastructure and organizational needs. Implementing your own solution, even with freeware software, is the most cost-intensive approach.

Server-side signature PKI

This is the newest concept in terms of PKI architecture. It implies using the existing organizational infrastructure in order to deploy a centralized document-signing application. This will be synchronized with a database that keeps record of all network users. Cost-wise, this might be the best solution (roughly $65 for each 3-year license).

2.5 PGP – a PKI alternative

Pretty good privacy is an encryption/decryption application used to transfer and validate data. PGP is often used to sign, encrypt and decrypt txt files, emails, folders and even logical hard-drive partitions. Message authentication and content integrity are the two main facilities of this type of application. The latter is used in order to establish whether a message has been altered since it was encrypted. The former is used to verify the sender's identity. Because the file content is encrypted, any modifications will lead to an erroneous decryption result. The RSA or DSA algorithms [1] are used for message signature. For this purpose, the PGP application computes a hash key(also called a message digest) and signs the document using both the hash value and the private key.

3. Problem solution – the Android/Web service approach

The presented solution is a proof-of-concept for the traditional PKI architecture which uses a mobile system to achieve .pdf-type document signing [2]. The system involves the existence of three components – the client (the owner of the .pdf document), the signee (an Android device capable of signing documents) and the CA/RA (which are both part of a single entity – a web service). The service can be adapted to use certificates acquired from a globally-recognized authority, but due to the fact that the costs are relatively high (for example, a Symantec certificate can reach 3000$ for a 2 year validity period), we use self-generated certificates.

3.1 Used technologies

The programming language of choice is Java. In order to make the development process easier, Eclipse and Netbeans IDEs (Integrated Development Environment) in order to create a graphical user interface, generate common method content (getters, setters, constructors), write event handlers. Communication between the client and the Android device is achieved via Bluetooth due to its availability and possibility to create a custom service which can later be discovered. All communication between the client and the web-service as well as the Android device and the web-service is achieved via SOAP request/response. These rely on WSDL (Web Service Description Language). WSDL is a XML based language that describes a series of methods that clients can invoke on the web-service. This is achieved by describing the web services as a series of network endpoints (or ports). A ports is defined by its association to a network

address. A collection of such ports define a web service. The SOAP messages are an abstract interpretation of exchanged data.
The digital certificates are based on the X.509 protocol. X.509 is an ITU-T protocol regarding PKI, CRL (Certificate Revocation List), certificate attributes and validation algorithms. Some of the attributes of a X.509 type certificate include version, serial number, used algorithm ID, authority, validity period, the public key, the signature and other optional fields. X.509 is not issue free, some problems have been encountered regarding the CRL size, root certificate revocation and using a blacklist concept for certificate validity (whitelist – trusted certificates – is recommended). Most internet browsers (such as Internet Explorer, Firefox, Opera, Chrome, Safari) include a preinstalled set of certificates, making it possible to acknowledge SSL certificates issued by well-known providers (Symantec, VeriSign etc.).
Another term that is related to the certificate concept is PKCS#12 (Public Key Cryptography Standards). This is an archiving format that allows storing multiple cryptographic objects in a single file. Usually, we use PKCS#12 for storing a digital certificate and the associated private key. This file may also be encrypted.
Java keytool [3] is a helper API included in the default JDK that helps generate and manage certificates. This allows users to generate their own public key/private key pairs, as well as the associated certificates for self-authentication and ensuring data integrity. Also, it allows users to keep local copies of public keys as certificates. These are stored in entities known as keystores. All entries in a keystore are uniquely identified via aliases (case-sensitive). The alias is generated when a new entry is added to the keystore. Each keytool command requires a keystore parameter. By default, keytool uses DSA for generating keys and SHA1withDSA for signature.
SHA1 is a cryptographic hash function that was designed by the United States National Security Agency and published by the United States. SHA is an abbreviation for "secure hash algorithm".
DSA is a standard issued by the United States Federal Government. It was proposed by the National Institute of Standards in 1991 and it is used in the Digital Signature Standard (DSS).
DSA key generation is a two-phase process – a choice of algorithm parameters and computing the private key for a single user.
In order to further speed up the development process, several third party Java APIs have been used.
Bluecove (also known as JSR-82) is a java library for Bluetooth that offers several useful features designated for working with Bluetooth adapters. Current Bluetooth stacks that Bluecove works with are WIDCOMM (Broadcom), Winsock (Microsoft) and Bluesoleil (IVT Corporation).
IText is another useful library that provides ease of use when working with PDF files. Features include PDF creation, digital signature with a wide array of customization options, content manipulation and so on. For digital signature purposes, IText uses a BouncyCastle provider.
BouncyCastle is a collection of APIs used in cryptography. Its architecture consists of light-weight APIs and JCE (Java Cryptography Environment) provider.
One particular issue is that Android already includes a built-in light version of BouncyCastle. Unfortunately, it lacks digital signature support. Since IText tries to access those classes, if we include the complete library we get classloader conflicts. To avoid this, a custom version of IText has been deployed. It uses a SpongyCastle provider, which is nothing more than the original version of BouncyCastle, but with small changes in terms of class names so that it runs smoothly on Android Operating Systems. However, the .jar file also requires a license in order to work. Such a license can be acquired or the user can request a 30-day trial license key (more information on this here: http://itextsupport.com/download/android.html.
JDBC (Java Database Connectivity) is a java-based data access technology. It was developed by the Oracle Corporation and

ISSN 2067 – 4074

allows Java programmers to define the way in which an application can use a relational database. JDBC connections allow the creation on execution of typical SQL statements, such as CREATE, INSERT, UPDATE and DELETE.
Radio Frequency Communication is a set of transport protocols implemented on top of the L2CAP protocol. It is also known as serial port emulation. RFCOMM provides a simple and reliable data stream to the user, highly similar to TCP. In the working example, RFCOMM is used as a transporter using OBEX over Bluetooth.
Service Discovery Protocol (SDP) allows devices to search for a particular service supported by other neighboring devices. A simple example of SDP is when we connect a headset to a mobile phone – SDP will determine if the headset supports headset profile, hands free profile, advanced audio distribution profile etc. Each service is assigned its own UUID (Universally Unique Identifier).

3.2 Debugging tools

For debugging purposes, there are several tools that were used during the development of this project. First off, NetBeans and Eclipse provide console output and a wide range of messages/exceptions to check step-by-step code execution. Also, Eclipse ADT (Android Development Toolkit) comes with an integrated emulator if no Android device is available. All Android devices come with a built-in USB Debugging option that allows the user to run the .apk image directly on the connected device. Most errors and stack backtraces may be investigated using the built-in Logcat feature. Then there is the issue of properly sending SOAP requests and responses. To ensure that the message format and payload is correct, we can either print the SOAP request/response messages or use Wireshark to listen on the device's NIC (Network Interface Card) and capture all http traffic. The latter solution is preferred simply because of the better control and information it provides the developer with.

3.3 Information workflow

Before we can talk about the steps needed to create a signature, let's take a look at the hardware/software prerequisites. First, we have to make sure we have a Bluetooth adapter available at both ends. If no default Bluetooth adapter is available, we can always use a USB Bluetooth dongle. We also need a working server that is reachable via LAN/Internet that will provide the signature verification services, the certificate generation, the user database administration etc. Finally, we need an Android phone.
The digital signature process can be broken down into a series of steps, as follows. Let's assume the client that wants to sign "sample.pdf" is "Alice", the Android phone that signs it is "Bob", and the server hosting the web services and the user database is "Charlie". First, "Bob" starts the application and logs in with user/password credentials. This will generate a SOAP request to the server. Here, "Charlie" will check these credentials against a user database. This database represents the list of users that are allowed to offer digital signature services. If "Charlie" determines that the credentials are valid, it will return a success message to "Bob". Now that we concluded that "Bob" is allowed to sign, he will have to check if it has a certificate file stored on the device (PKCS#12 format). If not, it will send a request to "Charlie". "Charlie" will invoke a .bat script to generate a new certificate using Keytool. It will keep a local copy and send "Bob" one via SOAP. "Bob" now meets all requirements, so he can turn on his Bluetooth device and start advertising a custom service via RFCOMM. Now "Alice" starts scanning the area for the desired service. If she finds the service it is looking for, it will push the .pdf file to "Bob" using OBEX_PUSH. "Bob" opens the file, sign it using the stored certificate and send the file back to "Alice" via email. "Alice" saves the file on her unit and sends a request to "Charlie" to check if the signature is valid. In order for "Charlie" to know which signee he is looking for, "Alice" has to know the name of the signee (this information will be

included in the email sent by "Bob" to "Alice"). "Charlie" will send a reply to "Alice" with a success or failure status, depending on the result of the certificate validation.
In this example, no separate entities were used for the certificate and the registration authority, the same machine/application handles both user management and certificate related operations. Furthermore, we consider a trusted base of users, meaning that the Android application does not allow remote registration. There is a given set of user/password combinations that are allowed to run the service and this will be decided by the system administrator.

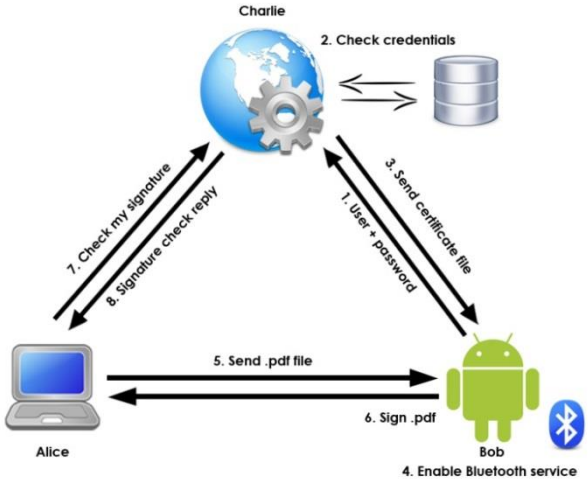

Figure 1 Solution Architecture

4. Conclusion

The mobile signature architecture comes with numerous advantages, such as flexibility, ease-of-use, lots of signature customization options, thanks to the iText library [4]. The initial deployment costs are close to zero given the users already own the necessary hardware (which is very likely, since all it requires is a minimum of two computers, one USB Bluetooth adapter and an Android OS smartphone). It could very well be integrated into companies that require some form of security implemented on their file transfers. Furthermore, public law signature offices could use a similar architecture in the nearby future in order to make their services more widespread and efficient in terms of administration, time and financial costs as well as service delivery speed improvements.
All tests have been run on an LG E730 Android device. The overall performance was fair, the actual digital signature on the .pdf document taking ~1 second. The rest of the process depends on the pace at which the system users issue commands (example: how fast the user can access his email inbox to retrieve the signed .pdf attachment). The application could be released and tested on more types of Android devices.
Since this system is developed having in mind corporations with a highly secure intranet, SOAP messages exchanged between the Android device and the other system components are not encrypted in the current version. In a more hostile environment, the Kerberos protocol comes to mind as a possible improvement on the system authentication mechanism.

References

[1] Wikipedia, DSA algorithm, https://en.wikipedia.org/wiki/Digital_Signature_Algorithm
[2] Catalin Boja, Data Security Solution for Mobile Applications, The 9th International Conference On Informatics in Economy – Education, Research & Business Technologies, Bucharest, May 7-8, 2009, ASE Publishing House, Bucharest, ISBN 978-606-505-172-2
[3] David Hook, Beginning Cryptography in Java, Wrox Press, 2005, ISBN:0764596330
[4] Bruno Lowagie, iText in Action, 2nd Edition, 2010, ISBN: 9781935182610
[5] Oracle, The Java EE 5 Tutorial, available at: http://docs.oracle.com/javaee/5/tutorial/doc/bnbhr.html
[6] Oracle, Java™ JDBC API, available at: http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/
[7] Mohan Atreya, Benjamin Hammond, Stephen Paine, *Digital Signature,* RSA Press, 2002
[8] Mark L. Murphy, *The Busy Coder's Guideto Advanced Android Developmen*, Commonsware, 2009.