Comparison Research on Digital Signature Algorithms in Mobile Web Services

Zuguang Xuan, Zhenjun Du, Rong Chen
School of Information Science and Technology
Dalian Maritime University
Dalian, China
xuanzuguang@hotmail.com

Abstract—**Mobile web services are playing an import role in the mobile information system. However, their security is a critical issue cumbering their application and development. The digital signature is the indispensable way to ensure the security of mobile web services and has great significance in practical applications. This paper studied the most popular digital signature algorithms such as DSA, RSA and ECDSA and compared these algorithms in theory. And the signature algorithms were implemented in Java ME on mobile device emulators, with various VM speeds simulated and different key sizes set. Experimental comparison results of the three signature algorithms were presented and analyzed. The results show that ECDSA is more suitable to generate the signature and RSA is more suitable to verify the signature on mobile devices. And a recommended solution to the digital signature in mobile web services is derived that it's to use the 1024-bit RSA scheme when the mobile client is required to verify the signature, and switch to adopt the 160-bit ECDSA scheme when the mobile client is required to become a signer.**

Keywords-Digital Signature; Mobile Web Service; Java ME

I. INTRODUCTION

The field of web services has expanded to include the mobile computing paradigm in constructing information systems. Handheld mobile devices participate in web services transactions through mobile web services. Mobile web services are typically mobile applications that consume traditional web services [1]. With the arrival of 3-G era, it's being a more and more important way to access web services using mobile devices. But, because of the characteristics of mobile devices, security is always a key factor affecting development of mobile web services.

HTTPS, SSL (Secure Socket Layer), and TLS (Transaction Layer Security) are connection-based security protocols. The basic idea is to secure communication channels and hence, secure everything that passes through those channels. Although HTTPS proves sufficient for most of today's Internet commerce applications, this approach has several problems: HTTPS can only provide point to point protection; all contents rather than one or several parts are encrypted; HTTPS is inflexible for applications that have special security and performance requirements. Mobile web services demand more flexible, customizable, and better-optimized security schemes.

XML (eXtensible Markup Language) is a main data exchange protocol in web services. The best method to secure web services is to bind XML file and security messages (eg. Digital Signature, Digest, Key, etc) together when clients communicate with the web services provider. XML Signature is a standard for providing message integrity. Within the world of web services, where plain text XML potentially transmitting sensitive data is whizzing between internet servers, having a mechanism for ensuring that messages are transmitted without modification is vital. Digital signature algorithms play a very important role in XML security [2] [3].

The main work of this paper is to compare the popular digital signature algorithms in theory and by experiment and to give some useful results to guide the application of digital signature algorithms in mobile web services. The remainder of the paper is organized as follows. In section 2, we analyze and compare three main types of digital signature algorithms in theory. In section 3, we make experiments to compare the performance of digital signature algorithms running on mobile devices. And section 4 concludes the paper.

II. SIGNATURE ALGORITHMS: RSA, DSA, ECDSA ANALYSES AND COMPARISON IN THEORY

A digital signature is a type of asymmetric cryptography. For messages sent through an insecure channel, a properly implemented digital signature gives the receiver reason to believe the message was sent by the claimed sender. Digital signatures are equivalent to traditional handwritten signatures in many respects; properly implemented digital signatures are more difficult to forge than the handwritten type. Digital signature schemes in the sense used here are cryptographically based, and must be implemented properly to be effective.

Digital signatures can guarantee message integrity and authenticity in an open network. To generate a signature, the sender first calculates a digest of his message using a hash function. Then he encrypts that digest with his private key to generate a signature. The receiver first decrypts the sender's signature into a digest using the sender's public key. Then the receiver calculates the digest from the sender's message. If the two digests match, the message is indeed from the sender and unaltered. The process of the signature generation and verification is as shown in Fig. 1.

This work has been partially supported by National Natural Science Foundation of China (No. 60775028), Dalian Science & Technology Program (No. 2007A14GX042) and Dalian Maritime University Youth Foundation (DLMU-ZL-200803).

978-1-4244-4639-1/09/$25.00 ©2009 IEEE

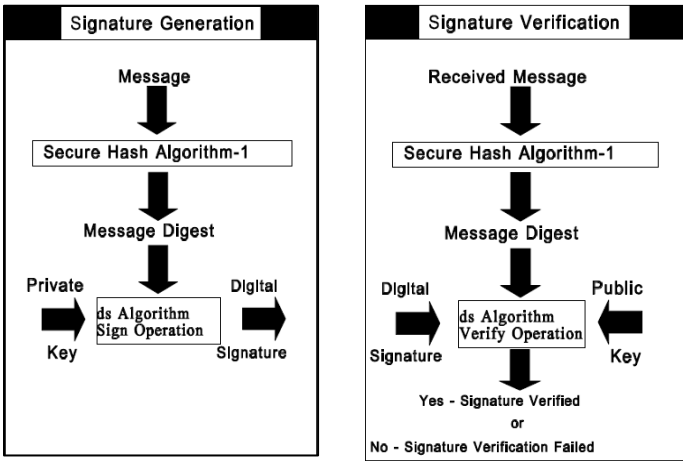

Figure 1. The process of signature generation and verification

There are three types of digital signature algorithms commonly used, such as RSA, DSA, ECDSA.

A. RSA

In cryptography, the RSA algorithm was publicly described in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT [4]; the letters RSA are the initials of their surnames, listed in the same order as on the paper. RSA is an algorithm for public-key cryptography. It is the first algorithm known to be suitable for signing as well as encryption, and one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys and the use of up-to-date implementations.

RSA, like most digital signature schemes, follows the "hash-then-sign" paradigm. The main steps are as follows:

1. Key generation

- Choose two distinct prime numbers p and q.
- Compute $n = pq$. And n is used as the modulus for both the public and private keys.
- Compute the totient: $\varphi(n) = (p-1)(q-1)$.
- Choose an integer e such that $1 < e < \varphi(n)$, and e and $\varphi(n)$ share no factors other than 1 (e.g. e and $\varphi(n)$ are coprime).
- Determine d (using modular arithmetic) which satisfies the congruence relation $de \equiv 1 (\mathrm{mod}\, \varphi(n))$.

The public key consists of the modulus n and the public exponent e. The private key consists of the modulus n and the private (or decryption) exponent d which must be kept secret.

2. Signing messages

- Produce a hash value h_1 of the message m: $h_1 = H(m)$.
- Produce a signature S using the private key *(d,n)* : $s = h_1^{\,d} (\mathrm{mod}\, n)$.

3. Verifying signature

- Get the public key *(e,n)* of the sender.
- Decrypt the signature S using the public key (e,n): $h = s^e (\mathrm{mod}\, n)$.
- Produce a hash value using the same hash algorithm in conjunction with the public key of the sender and compare it with the result h .If the two agree, the signature is valid, otherwise, the signature is invalid.

RSA-PSS is a new signature scheme that is based on the RSA cryptosystem and provides increased security assurance. It was added in version 2.1 of PKCS #1. In this paper, we choose RSA-PSS to test the performance of the RSA.

B. DSA

The DSA (Digital Signature Algorithm) is a United States Federal Government standard for digital signatures. It was proposed by the National Institute of Standards and Technology (NIST) in August 1991 for use in their Digital Signature Standard (DSS). DSA is for signatures only and is not an encryption algorithm [5].

The main steps of DSA are as follows:

1. Choice of algorithm parameters:

Choose a cryptographic hash function *H*, a key size *L,* a prime *q* with the same number of bits as the output of *H,* a *L*-bit prime *p* such that *p*–1 is a multiple of *q,* a number *g* whose multiplicative order module *p* is *q*.

2. Key generation

- Choose *x* by some random method, where $0 < x < q$.
- Calculate $y = g^x \bmod p$.
- Public key is (*p*, *q*, g, *y*). Private key is *x*.

3. Signature generation

- Generate a random message value *k* where $0 < k < q$.
- Calculate $r = (g^k \bmod p) \bmod q$.
- Calculate $s = (k^{-1}(H(m) + x*r)) \bmod q$, Where *H* is the hashing function and *m* is the message.
- Recalculate the signature in the unlikely case that *r*=0 or *s*=0.
- The signature is (*r*,*s*).

4. Verifying signature

- Reject the signature if either $0 < r < q$ or $0 < s < q$ is not satisfied.
- Calculate $w = (s)^{-1} \bmod q$.
- Calculate $u1 = (H(m)*w) \bmod q$.
- Calculate $u2 = (r*w) \bmod q$.
- Calculate $v = ((g^{u1}*y^{u2}) \bmod p) \bmod q$.

- The signature is valid if $v = r$.

C. ECDSA

The ECDSA is a signature scheme with appendix based on ECC (Elliptic Curve Cryptograph) [5]. ECDSA was first proposed in ANSI X9.62, and it's an elliptic curve analog of DSA. It is designed to be existentially unforgeable, even in the presence of an adversary capable of launching chosen-message attacks. The main steps of ECDSA are as follows.

1. Signature generation algorithm

- Initially, the curve parameters (q,FR,a,b,G,n,h) must be established at the desired security level.
- Produce a key pair suitable for elliptic curve cryptography, consisting of a private key d_A (a randomly selected integer in the interval $[1,n-1]$) and a public key Q_A (where $Q_A = d_A G$).
- Calculate e = HASH(m), where HASH is a cryptographic hash function, such as SHA-1.
- Select a random integer k from $[1,n-1]$.
- Calculate $r = x_1 (\mathrm{mod}\ n)$, where $(x_1,y_1) = kG$. If $r = 0$, select a different random integer k from $[1,n-1]$ again.
- Calculate $s = k^{-1}(e + rd_A)(\mathrm{mod}\ n)$. If $s = 0$, select a different random integer k from $[1,n-1]$ again.
- The signature is the pair (r,s).

2. Signature verification algorithm

- Receive the signature (r,s) and the public key Q_A of the sender .
- Verify that r and s are integers in $[1,n-1]$. If not, the signature is invalid.
- Calculate e = HASH(m), where HASH is the same function used in the signature generation.
- Calculate $w = s^{-1} (\mathrm{mod}\ n)$.
- Calculate $u_1 = ew (\mathrm{mod}\ n)$ and $u_2 = rw (\mathrm{mod}\ n)$. The signature is valid if $r = x_1 (\mathrm{mod}\ n)$, invalid otherwise.

D. Comparison of The Signature Algrorithms in Theory

RSA, DSA and ECC are popular public-key algorithms in use. ECDSA algorithm based on elliptic curve discrete logarithm problem is the performance-improved DSA and it has overall advantages over traditional DSA. RSA algorithm based on big integer factorization problem features with the simple math principle and easy to implement in engineering applications, but its security unit intensity is relatively low. Mathematical theory of ECDSA algorithm is more complex and is more difficult to implement, but its security unit intensity is relatively high. While sub-exponential algorithms can solve the integer factorization problem for RSA, only exponential algorithms are known for the ECC algorithm problem. This allows ECC to achieve the same level of security with smaller key sizes and higher computational efficiency; ECC-160 provides comparable security to RSA-1024 and ECC-210 provides comparable security to RSA-2048. The comparison of RSA, DSA and ECC are as follows in Table Ⅰ.

TABLE I. THE COMPARISON OF RSA AND ECC

Decipher time (MIPS Years)	RSA/DSA key size (bits)	ECC key size (bits)	RSA/ECC The proportion of key size
10^4	512	102	5:1
10^8	768	136	6:1
10^{11}	1024	160	7:1
10^{20}	2048	210	10:1
10^{78}	21000	600	35:1

Although ECDSA has theoretical advantages over RSA and DSA, yet for the application in mobile web services, the limited performance of mobile devices must be carefully considered. Experiments on various devices are made in the following section to show a suitable solution.

III. EXPERIMENTAL COMPARISON

Experiments are made in this section, with programs written in Java ME (Java Micro Edition). Java ME provides a robust, flexible and platform-independent environment for applications running on mobile and other embedded devices, widely used in mobile systems [7].

In our experiments, the running performances of DSA, RSA and ECDSA are tested on WTK (Sun Java Wireless Toolkit) 2.5.2 platform. Also, Bouncy Castle Crypto package [8] is imported in the experiments，which is a implementation of cryptographic algorithms for all Java platforms (including Java ME).

We choose the same algorithm SHA-1 to generate message digest in the process of DSA, ECDSA and RSA signature, and the time of generating digest is also the same, so we just test the time of key generation, signature generation and signature verification separately.

The device emulator is provided in WTK, which can simulate the constrained environment of a real device. Although the emulator does not represent a real device, adjusting the performance settings can give us useful information about how our application performs under varying runtime conditions just like running on different devices.

To simulate the actual speeds of various real devices, enable VM speed emulation function in WTK and choose the emulation speeds from 100-1000 bytecodes/millisecond as shown in Fig. 2.

Figure 2. Adjusting the mulator's performance

The variation of storage size and heap size (both from 512kB-65536kB) almost have no impact on our experiment due to the small size of our program, so we choose the default size (set blank) as shown in Fig. 3.

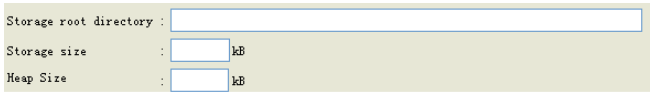

Figure 3. Storage preferences

First, the key sizes of DSA, RSA and ECDSA are separately set as 1024/1024/160, and the VM Speed is adjusted. The results of the experiment are as shown in Table Ⅱ (only the results for the speeds 350 and 1000 are listed here). Table Ⅱ shows the VM Speed has an important impact on signature time. The performance of the device is also a key factor of improving its security.

TABLE II. COMPARISON OF SIGNATURE TIME FOR VARIOUS VM SPEEDS

VM Speed: (bytecodes/millisecond)		**350**	**1000**
Key Generation Time (ms)	***DSA***	17636355	5869522
	ECDSA	370093	85150
	RSA	3886422	1272500
Signature Generation Time (ms)	***DSA***	47531	15171
	ECDSA	385320	91800
	RSA	76385	25755
Signature Verification Time (ms)	***DSA***	100901	33602
	ECDSA	483356	114450
	RSA	5848	1950

Secondly, we set the VM Speed as 1000 and adjust the key sizes. The results of the experiment are as shown in Table Ⅲ. The results for the key size of 2048/2048/210 for DSA/RSA/ECDSA are not listed in Table Ⅲ, because the running time is too long, which shows this key size is impractical to use on mobile devices at all. Table Ⅲ shows when the key size is smaller, the running time is shorter, while the safety level is on the decrease.

TABLE III. COMPARISON OF SIGNATURE TIME FOR VARIOUS KEY SIZES

Key Sizes (bits)		**Key Generation Time (ms)**	**Signature Generation Time (ms)**	**Signature Verification Time (ms)**
DSA	**768**	3307048	9751	19922
	1024	5869522	15171	33602
ECDSA	**132**	70702	70722	91811
	160	85149	91801	114453
RSA	**768**	609755	12103	1146
	1024	1262500	25755	1950

We can see from the tables that the key size of 1024/1024/160 for DSA/RSA/ECDSA is the optimal choice with a compromise of safety and running time. In this key size, the sum of the key generation time and the signature generation time for ECDSA is shorter than DSA and RSA, but its signature verification time is the longest. The RSA signature verification time is the shortest but its key generation is timing consuming. So 1024-bit RSA is recommended here for signature verification on the mobile device, and 160-bit ECDSA is recommended for generating keys and signatures on the mobile device. But we should take this in mind that the sum of the key and signature generation time of 160-bit ECDSA is still too long (about 3 minutes in this experiment), the key and signature generation should run in a separate background thread so as not to block the application running and further optimized measures are required to improve the running performance.

So, for the practical use of the digital signature in mobile web services, the suitable signature solution, we think, is to choose the 1024-bit RSA scheme when mobile clients need to verify the signature and use the 160-bit ECDSA scheme when the mobile client needs to be the signer. Or, if the performance of the mobile device is relatively low, the mobile client may only be a RSA signature verifier and cannot be required to be a signer.

IV. CONCLUSIONS

This paper analyzes and studies digital signature algorithms such as DSA, RSA and ECDSA in the mobile web service field. First, we study these algorithms in theory, then implement them on device emulators, and finally we analyze and compare the experiment results and draw our conclusions on a solution to the digital signature in mobile web services that 1024-bit RSA is recommended for signature verification on mobile devices, and 160-bit ECDSA is recommended for generating keys and signatures on the device in mobile information systems. But there is still a great deal of research work on the digital signature to do. Our future work is to further improve the performance of digital signature algorithms and optimize the safety architecture of the mobile information system.

REFERENCES

[1] Sun Microsystems, Java 2 Micro Edition Web Service Specification version 1.0, http://jcp.org/jsr/detail/172.jsp, October, 2003.

[2] B. Dournaee, XML Security,1st ed., Emeryville, CA: McGraw-Hill Osborne Media, 2002.

[3] V. Engelen, A. Robert and W. Zhang , "An Overview and Evaluation of Web Services Security Performance Optimizations," Proceedings of the IEEE International Conference on Web Services, ICWS 2008, pp 137-144, 2008.

[4] RSA Laboratories, PKCS #1 v2.1: RSA Cryptography Standard, June, 2002.

[5] A. Menezes, P. V. Oorschot, and S. Vanstone, Handbook of Applied Cryptography, 1st ed., Boca Raton, FL: CRC Press, 2001.

[6] Standards for Efficient Cryptography Group, SEC 1: Elliptic Curve Cryptography, September 2000.

[7] M. J. Yuan, Enterprise J2ME: Developing Mobile Java Applications. Upper Saddle River, NJ: Prentice Hall, 2003.

[8] Bouncycastle official website, http://www.bouncycastle.org/.