Digital identity-based multisignature scheme implementation

Francisco Javier Buenasmañanas Domíguez,
Ascensión Hernández Encinas, Araceli Queiruga Dios
Dept. Applied Mathematics
University of Salamanca
Salamanca, Spain
Email: {u61352, ascen, queirugadios}@usal.es

Luis Hernández Encinas
Dept. of Information Processing and Coding
Applied Physics Institute, CSIC
Madrid, Spain
Email: luis@iec.csic.es

***Abstract*—Digital signature, as an official signature, have many applications in information security, including authentication, data integrity, and non-repudiation. When a private or public document must be signed by a group of people, we call it multisignature scheme if all and every single member of the group signs the document.**

An identity-based digital multisignature is a multi signer digital signature so that the multiple private keys are generated by a trusted third part from signer's identities. In this paper, an efficient Java implementation to a recent identity-based multisignature scheme based on RSA is proposed.

***Keywords*-RSA; digital signature; multi-signature scheme; Java.**

I. INTRODUCTION

Adi Shamir [1] introduced a novel type of cryptographic scheme, based on the identity of the users, which enables any pair of users to communicate and sign documents securely. Moreover, it is possible to verify each other's signatures without exchanging private or public keys, without keeping key directories, and without using the services of a third party. The scheme assumes the existence of trusted Key Generator Center (KGC), with the role of giving each user a personalized smart card when he first joins the network. The card contains the secret key, and programs for message encryption/decryption and signature generation/verification. The user chooses any combination of name, social security number, e-mail or telephone number as his public key.

The scheme is adequate for closed groups of executives of a multinational company or the branches of a large bank or members of a sports club since the headquarters of the corporation can be the KGC that everyone trusts. This scheme can be the basis for a new type of personal identification card with which everyone can digitally sign checks, legal documents, and be electronically identify.

Moreover, a digital multisignature is a digital signature of a message or document generated by multiple signers with different private keys [2].

An identity-based digital multisignature, based on Shamir's identity-based scheme, is a digital signature of a message generated by multiple signers that obtains their private keys from a KGC, and the public keys from their own identities. Such practical and secure multisignature schemes were proposed to be applied for mobile communications [3].

In 2008, Harn and Ren [4] proposed an efficient identity-based RSA multisignature scheme and it seams to be secure against known attacks like forgerability under chosen-message attack, multi-signer collusion attack and adaptive chosen-ID attack. Authors propose the scheme with the most important multisignature properties: the length of the multisignature was fix and the verification time was also fix, regardless of the number of signers.

Some flaws on Harn-Ren identity-based RSA multisignature scheme were published a year later [5]. These drawbacks led to the proposal of a new system two years later. In fact, in [6] two security loopholes were discovered in Harn-Ren scheme and a new one was proposed. The resultant protocol was suitable for wireless communications because it is not only possessing security but also saving computation resources and communication bandwidth.

The implementation proposed in this paper supposes an efficient Java implementation of the improved identity-based multisignature scheme based on RSA suggested in [6]. This paper is organized as follows. Section II will detail the RSA cryptosystem and a short overview of existing identity-based multisignature schemes. Some attempts of using Matlab to implement these cryptosystems and the efficient Java implementation will be shown in Section III. Finally, conclusions and future works will be presented in Section IV.

II. RELATED WORKS

In this section, we make a brief overview to RSA cryptosystem as well as Shamir's identity-based signature scheme. Harn-Ren's scheme as well as the improved multisignature scheme will be reviewed.

A. RSA cryptosystem

RSA cryptosystem [7] consists of three phases: key generation, encryption and decryption.

1) Key generation for a user U:

a) Select two large prime numbers p, q and computes $n = p \cdot q$ and $\phi(n) = (p-1)(q-1)$.

b) Select a positive integer e, $1 < e < \phi(n)$, such that $\gcd(e, \phi(n)) = 1$.
c) Compute the inverse of e in $Z^*_{\phi(n)}$, d, so that $e \cdot d \equiv 1(\text{mod } \phi(n))$.

The public key of U is the pair (n, e) and his private key is d. For security reasons, the values p, q and $\phi(n)$ must be kept secret.

2) Encryption process: If user B, wishes to cipher the message, M, and send it to user A, he carries out the following operations:

a) He obtains A's public key: (n_A, e_A).
b) He represents M as an integer in the range $[0, n_A - 1]$, even splitting m into smaller blocks if necessary.
c) The ciphered message is $c = M^{e_A}(\text{mod } n_A)$.

3) Decryption process: To decipher the cryptogram c and recover the original message, M, user A simply uses his private key d_A and computes $c^{d_A} \equiv M^{e_A d_A} \equiv M(\text{mod } n_A)$.

Asymmetric-key cryptosystems allow the sender to digitally sign a message, so that the receiver can check that the message is authentic and not modified.

Suppose that A, wishes to digitally sign a public document, M, and send it to B. The steps are the following.

a) The first step is to apply a *hash function* to the document, creating the document digest [8], $H(M) = m$, and encrypts it using his private key: $r \equiv m^{d_A}(\text{mod } n_A)$.
b) He ciphers the value r with B's public key to obtain the signature $s \equiv r^{e_B}(\text{mod } n_B)$.

Once the document and the signature are received by B from A, he can perform the verification phase as follows:

a) He computes $s^{d_B}(\text{mod } n_B) \equiv r^{e_B d_B}(\text{mod } n_B) \equiv r$.
b) He determines $r^{e_A}(\text{mod } n_A) \equiv m^{d_A e_A}(\text{mod } n_A) \equiv m$.
c) He deciphers c to obtain M, and checks whether the hash of M, $H(M)$, matches m. If it does, the signature is valid.

The security of the encryption and decryption processes, and the digital signature scheme based on RSA, depend on the difficulty of solving the factorization problem, which at present is considered computationally infeasible.

B. Shamir's identity-based signature scheme

First of all, each signer completes his registration with KGC, and KGC generates the signer's secret key using their own identities. On the other hand, the signature's public verification key is the signer's identification. This scheme reduces the costs of verifying the public key. The process is divided into the following phases:

1) KGC keys:

a) KGC picks two large primes p, and q, to compute n and $\phi(n)$.
b) Chooses a random public key e, satisfying §II-A conditions.

2) Signer secret key generation phase:

a) Signer j sends individual information and his identity i_j to KGC for registration.
b) After KGC accepts the user's identity, KGC uses his private key d to create a secret key $d_j \equiv i_j^d(\text{mod } n)$ from signer's identity i_j. Subsequently, d_j will be sent back to the signer as his secret key.

3) Signing phase: In the process of signing a document or message digest m, the signer uses his secret key d_j and the public key e of the KGC to produce the signature $\sigma = (t, s)$. The signing process is as follows:

Signers choose a random number r to compute $t = r^e \text{mod } n$. The secret key d is used to compute $s = d \cdot r^{H(t,m)}(\text{mod } n)$. Then, (t, s, m) is transmitted to the receiver, and $\sigma = (t, s)$ is the signature of the message.

4) Verification phase: When the receiver receives the signature $\sigma = (t, s)$ for the message m from signer i_j, the public key e of the KGC and signer's identity i_j can be used to verify the validity of the signature:

$$s^e \equiv i_j \cdot t^{H(t,M)}(\text{mod } n).$$

C. Harn-Ren efficient identity-based RSA multisignature

1) Private key generation phase: In this algorithm, every signer obtains his private key from the KGC:

a) Every signer sends their individual information to the KGC for registration.
b) KGC, with his private key d and the message digest of identity i_j, generates the private key $d_j \equiv i_j^d(\text{mod } n)$ of i_j signer.

2) Signing phase: To generate an identity-based digital multisignature every i_j signer from a group of signers, $i_1, i_2, \ldots, i_l$, follows these steps:

a) Chooses a random integer r_j, and with his public key, e, computes
$$t_j \equiv {r_j}^e(\text{mod } n).$$
b) Broadcasts r_j to all signers.
c) After receiving r_j, $j = 1, 2, \ldots, l$, each signer computes
$$t \equiv \prod_{j=1}^{l} r_j(\text{mod } n), \text{and } s_j \equiv d_j \cdot r_j^{H(t,M)}(\text{mod } n).$$
d) Broadcast s_j to all signers.
e) After every signer has received s_j, $j = 1, 2, \ldots, l$ from the others, compute s as
$$s \equiv \prod_{j=1}^{l} s_j(\text{mod } n).$$

The multisignature of a message m is $\sigma = (t, s)$. In this scheme every signer's signature is the same as Shamir's scheme.

3) Verification Phase: To verify the multisignature $\sigma = (t, s)$ made by signers with identities $i_1, i_2, \ldots, i_l$ of m,

$$s^e \equiv (i_1 \cdot i_2 \cdots i_l) \cdot t^{H(t,m)} (\text{mod } n).$$

If this verification equation is successful, then the information has a legitimate signature.

Harn-Ren's multisignature scheme does not protect the signer's signature secret key from being exposed [6]. Anyone is able to obtain the signer's secret key d_j using broadcast data (r_j, s_j) and signature (m, σ). Moreover, if e is a small value, an attacker is able to obtain the signer's secret key d through the public information (e, m, s) .

D. The improved authentication scheme

To avoid the two mentioned loopholes in Harn-Ren multisignature scheme, a new one was proposed in [6], where the KGC keys phase and signer secret key generation phase is the same as the original scheme in §II-C.

1) Signing phase: As before, if the group of signers $i_1, i_2, \ldots, i_l$ want to jointly sign the document m, each signer j performs the same steps mentioned in §II-C, except that now the values s_j are defined as:

$$s_j \equiv d_j^t \cdot r_j^{H(t,M)} (\text{mod } n).$$

2) Verification phase: When the receiver receives multisignature message (m, σ) , the public key e of the KGC and the identities of all the signers $i_1, i_2, \ldots, i_l$ can be used to verify the validity of the signature. The verification formula is as follows:

$$s^e \equiv (i_1 \cdot i_2 \cdots i_l)^t \cdot t^{H(t,m)} (\text{mod } n).$$

If verification is successful, then the information has a legitimate signature. Otherwise, it is an illegal signature.

III. Implementation and procedures

We have developed the Harn-Ren improved multisignature scheme. We have started with Matlab, with the use of functions and toolboxes to encrypt, decrypt and sign messages with RSA cryptosystem, with real parameters, and we changed to Java to code a more efficient programm.

A. Matlab and big integers

To implement RSA cryptosystem we need to find big integers, at least 1024 or 2048 bits keys, to be sure that RSA is secure against known attacks. Trying to work with Matlab, we found a toolbox, `vpi`, to compute variable precision arithmetic operations.

First of all we started the cryptosystem implementation trying to encrypt and decrypt a short message to check Matlab possibilities. We took parameters p and q with length of about 155 digits. In this case, the calculation of public and private key is fast, and also the calculation of the encrypted message, but to get the plain text is very slow, because the modular power with Matlab is not enough efficient. This was the reason to change to Java language.

B. Java implementation

We have developed a Java a digital identity multisignature application. Although Java is object oriented, being a simple and algorithmic implementation, we have divided the program into two classes, inside the `multisignatures_identities` package. The first class is called `Identities`, this class contains the embodiment of the signature and verification. The second class name is `Hash`, and performs a hash function from a string and returns the string's digest.

`Identity` class is composed by the following attributes:

1) RSA parameters: `p, q, n, fi, d, e`.
2) Number of participants: `num`
3) Each participant has: Identity (`i_j`), Private key (`d_j`), and other data used in the multisignature (`r_j, t_j, s_j`).
4) The values of the multisignature: `s, t`.

`Identity` class contains the methods that carry out the following actions:

1) Calculation of RSA parameters:
   a) `calculate_module`: returns the module `n` when `p, q` are known.
   b) `calculate_fi_euler`: returns `fi` when `p, q` are known.
   c) `calculate_d`: returns private key `d`, when public key and `fi` are known.
2) Calculation of identities and private keys:
   a) `calculate_identities_and_private _keys`: with the number of participants `num`, the private key `d`, and the modulus `n`, the identities `i_j` and private keys for each signer `d_j` can be calculated.
3) Other estimates:
   a) `calculate_first_step`: with the number of participants `num`, the public key `e` and the modulus `n`, values `t_j` and `r_j` can be obtained.
   b) `calculate_third_step`: calculate the value `t` with some of the previously calculated parameters.
   c) `calculate_s`: obtains the value `s` with some of the previously calculated parameters.
4) Some views on screen:
   a) `initial_parameters_view`: to show the following parameters: `p, q, n, fi, d, e`.
   b) `identities_and_private_keys_view`: to show the following parameters for each participant: `i_j, d_j`.
   c) `first_step_values_view`: to show `t_j, r_j` parameters for each participant.
   d) `third_step_values_view`: to show `t, s_j` parameters for each participant.
5) Signature verification:

a) `signature_verification`: verifies the returned signature with a boolean value: true if the signature is verified or false if not.

`Hash` class is composed by a serie of attributes:

1) `md`: is `typeMessageDigest`, where the hash function is of type SHA-1.
2) `buffer`: is an array of bytes, which contains the string to calculate the hash.
3) `digest`: byte that includes the string for conversion.
4) `hast`: the string which will store the hash value.

`Hash` class includes the following methods:

1) `Hash`: is the constructor method.
2) `getHash`: will calculate the hash for a string.

To develop the proposed multisignature scheme, we have used two classes: `BigInteger` and `BigDecimal`. These classes' types have advantages over the types primitive. When big numbers are needed in Java, the best option is to use these classes. In fact, their storage limit is the same limit as the Java virtual machine memory limit.

The `BigDecimal` class is only used to generate random numbers with the `random()` method of the Math library. The `BigInteger` class was more useful to the program because of some of the methods provided by this class. The methods that were interesting for us were:

1) `multiply(BigInteger val)`: it returns the multiplication of this `BigInteger` with the input parameter.
2) `subtract(BigInteger val)`: it returns the subtraction of this `BigInteger` with the input value.
3) `mod(BigInteger m)`: it returns the value of `BigInteger` module `m`, with `m` the input value.
4) `modInverse(BigInteger m)`: it returns the inverse of this `BigInteger` module `m`.
5) `modPow(BigInteger exponent, BigInteger m)`: it returns the pow of this `BigInteger` with exponent `m`.
6) `compareTo(BigInteger val)`: it compares `BigInteger` with the parameter passed in the method and return `0` if they are equal.

C. Benefits from this developments

We have calculated the CPU time to perform an identity based multisignature and the time to verify the multisignature with the proposed Java implementation. We have used a `System` class method called `currentTimeMillis()`. This method returns the current time in milliseconds. The needed average time to multisign a document by 10 signers is 88.3ms, and 1.3ms to verify. If we take 100 signers, the time to multisign the same document is 636ms and 3.2ms to verify.

These are two benefits related to the development:

1) The time to sign and verify a document is slow.
2) The possibilities offered by java environment are good. The source code detailed in section §III-B could be added to a java card applet to get a secure environment that allows different people to multising documents.

IV. Conclusion and future work

As we presented, some identity-based identification and signature schemes have been implemented using Java, as can be shown in [9], but there is no implementation related to an identity-based multisignature scheme based on RSA. We studied the possibilities of a software like Matlab, but we recognize that it does not work properly with big integers, that are needed to encrypt, decrypt and sign messages with RSA, and to multisign messages or documents with some users, the calculations are more slowly that the case of single RSA.

We have chosen the Java programming language because of its efficiency and because we are developing some Java Card applets that enable to digital sign documents.

Acknowledgment

This work has been supported by Fundación Memoria D. Samuel Solórzano Barruso under the project FS/7-2010.

References

[1] A. Shamir, "Identity-based cryptosystems and signature schemes", Advances in Cryptology (Crypto'84), vol. 196, 1984, pp. 47–53.

[2] R. Durán Díaz, F. Hernández Álvarez, L. Hernández Encinas, and A. Queiruga Dios, "A review of multisignatures based on RSA", Proceedings of The 4th International Information Security & Cryptology Conference (ISCTURKEY'10), 38–44. Ankara (Turkey), May 2010.

[3] Y.F. Chang, P.C. Chen, and T.H. Chen, "A Verifiable Identity-based RSA Multisignature Scheme for Mobile Communications," Journal of Computers, vol. 20, 3, 2009, pp. 3–8.

[4] L. Harn and J. Ren, "Efficient identity-based RSA multi-signatures", Computers & Security, vol. 27, 2008, pp. 12–15.

[5] Y.F. Chang, Y.C. Lai, and M.Y. Chen, "Further Remarks on Identity-based RSA Multisignature," PRoc. Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 2009, doi:10.1109/IIH-MSP.2009.137.

[6] F.Y. Yang, J.H. Lo, and C.M. Liao, "Improvement of an Efficient ID-Based RSA Multisignature," International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), 2010, pp. 822–826.

[7] R.L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Commun. ACM, vol. 21, 1978, pp. 120–126.

[8] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of applied cryptography, CRC Press, Boca Raton, FL, 1997.

[9] S.Y Tan, S.H. Heng, B.M. Goi, and J.J. Chin and S. Moon, "Java Implementation for Identity-Based Identification," International Journal of Cryptology Research, vol. 1, 1, 2009, pp. 21–32.