Secure digital signature using RSA

Wendy H. Chun
Department of Computer Science,
University of California, Santa Barbara, CA 93106
Email: chun0216@cs.ucsb.edu
June 9, 2010

Abstract— **As the Internet became part of our lives, we use Internet to make our lives convenient. For example, now, we tend to send an electronic mail instead of hand-written mail due to the speed and convenience. However, when we receive an electronic mail, how can we verify that whether it is sent by right person or not? As a result, everyday, we live in the insecure world where our identities are used by other people without even acknowledged about that. In order to prevent this, digital signature is a cryptographic scheme that is used to reinforce the security by authentication. In digital signature, there are many cryptographic algorithms, such as RSA, Elliptic curve, and ElGamal, are used for signing and verifying. In this paper, we will cover the whole process of the digital signature authentication using RSA, that is one of the oldest public key systems algorithm.**

I. Introduction

Digital signature is similar as the hand-written signature that we put for verification everyday. Sender sends a data with digital signature and when recipient receives it, the recipient verifies the sender by the digital signature that is sent it with the message. Digital signature can be created by many different cryptographic algorithms, such as Elliptical Curve, ElGamal, or other public-key cryptography techniques, but in this paper, we will create it by applying the oldest methods, RSA. Before we talk about the RSA algorithm, digital signature scheme is first described by Whitfield Diffie and Martin Hellman in 1976 using Diffie-Hellman algorithm [1]. However, it was not a strong algorithm because all it does is key exchange so soon after the Diffie-Hellman algorithm, RSA algorithm is invented by Ronald Rivest, Adi Shamir, and Len Adleman, which does not only key exchange but also encryption and decryption, and till now, the algorithm is popularly used for digital signature, even if there are other public-key cryptography techniques available.

In this paper, at first, in section II, we will talk about hash functions to create a message digestion and in section III and IV, we will talk about standard RSA algorithm for encryption and decryption. After that, in section V, we will talk about the problem of digital signature created by standard RSA algorithm and also solution in order to make more secure.

II. Message Digestion

When we look at the length of actual hand-written signature, it is fairly short because the purpose of signature is for quick verification. Similarly in cryptography, we want to keep the digital signature as short but yet important as possible and using a message digestion algorithm, we can reduce the message input into fixed length of pseudo-random output [2]. In order to create a message digestion, we need to apply one of the hash functions to the message and MD2, MD5, and SHA-1 are hash functions that are popularly used in nowadays. Because SHA-1 is known to be the most frequently used function because there are several problems in MD2 and MD5 such as speed, collision, and insecureness [3], we will use SHA-1 algorithm in this paper to generate a message digestion. Table 1 shows the comparison of speed by different hash functions [4].

Table 1: Parameter for special hash function.

hash function	block length	relative speed
MD4	128	1.00
MD5	128	0.68
RIPEMD-128	128	0.39
SHA-1	160	0.28
RIPEMD-160	160	0.24

A. Secure Hash Algorithm

In this section, we will show how using SHA-1 algorithm can make the message into fixed short length message. First, we have a message M that has a length at most $2^{64} - 1$ and we append this with 1, that is $M' \leftarrow M \circ 1$. Then we append M' with minimum number of zeros to make the length of M' becomes multiple of 448 that is, $|M'| = k \times 512 - 64$. After that, we add another 64 bits of extended original message M using big-endian and thus, the final length of M' will become multiple of 512. After running the M' through SHA-1 algorithm, we get the value of H_0, H_1, H_2, H_3, and H_4 in which each value is 32 bit words and finally, at the end, we concatenate those five hash values, $SHA1(x) = H_0H_1H_2H_3H_4$, which results 160 bits long. Because SHA-1 returns 160 bits long string regardless of the length of original message we use, this shows that our message of digestion is always going to be 160 bits long, which is pretty short.

III. Key generation

Before we encrypt the message, we also need to create keys that we are going to use. First, we pick two odd prime numbers for p and q that are randomly chosen, then we assign n as the product of those two numbers, $n = p \times q$. Using this n, a public key exponent is chosen in $1 < e < \varphi(n)$ range. $\varphi(n)$ is an *Euler function* and according to the theorem in [4], *if n is a prime number then* $\varphi(n) = n - 1$. Using this theorem, we can compute $\varphi(n)$ as $\varphi(n) = \varphi(p \times$

$q) = \varphi(p) \times \varphi(q) = (p-1) \times (q-1)$. Thus, public key exponent e can be chosen between $1 < e < (p-1) \times (q-1)$ and also satisfy $gcd(e, \varphi(n)) = 1$ condition. After that, we choose the private key exponent d using same range, $1 < d < (p-1) \times (q-1)$ and also satisfy $d \times e \equiv 1 mod(p-1) \times (q-1)$ condition. In this case, because we already know all other values, e, p, and q, we can easily compute d using *Extended euclidean algorithm* [4].

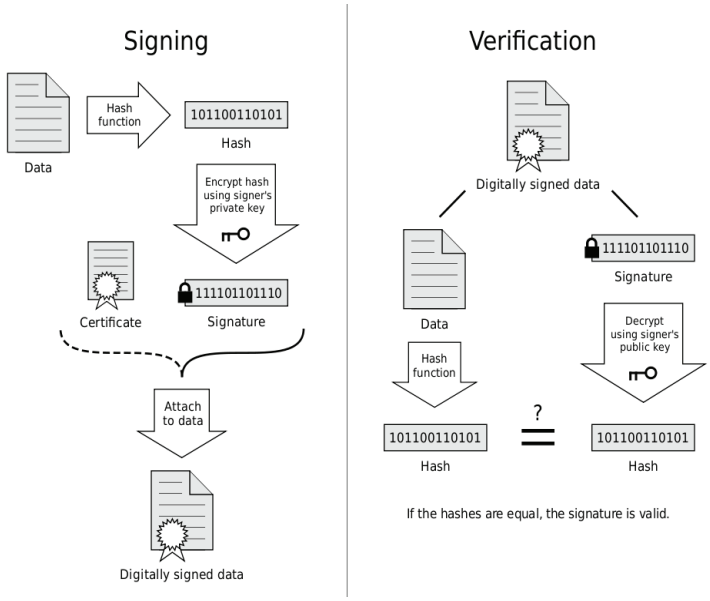
IV. Sign and verification

After creating a message digestion and keys, we use standard RSA algorithm to create signature and later to verify the signature [5]. First, let encryption and decryption functions as:

$$s(m) = m^d \ mod \ n$$
$$m(s) = s^e \ mod \ n$$

The signing process is very simple by using d, sender's private key, as an exponent to the message digestion. Then we attach this signature s with original message m and send it. During verification proces, when recipient receives the message and the signature, he or she first uses sender's public key to the signature and get the message, say M'. Because M' is created by hash function SHA-1 and it is infeasible to invert it back to M, the recipient should apply hash function to the original message M instead and then compare that with M' we got. In other words, if M' and $H(M)$ is the same, the signature is successfully verified; otherwise, not. The figure below shows the process of signing and verification.

Figure 1: Signing and verification process



V. Problem and solution

Suppose that we have a low public key exponent e, for example when $e = 3$, and small m, the message can be easily breakable in polynomial time by taking e^{th} root because RSA algorithm is a *deterministic encryption algorithm* in which there are no such random components [6]. In order to prevent this flaw, we use encryption padding schemes to increase the size of message so that it is hard to break, and/or also signature scheme to improve probabilistic signature. As a result of using these schemes, it makes the RSA algorithm not only becomes more secure but also makes deterministic to probabilistic scheme [7]. Here are two schemes we can apply to create a stronger signature.

A. Optimal Asymmetric Encryption Padding

Using an OAEP, Optimal Asymmetric Encryption Padding, we first convert m into padded m and then use it in normal signing equation, $s = (m_{padded})^d \ mod \ n$. OAEP is based on Feistel network [8] to randomize ciphertext in certain public-key encryption scheme. First, we choose two integers, k_0 and k_1 and also let the length of message as n. According to the definition, the length of M, the length of plaintext, should be $|M| = n - |k_0| - |k_1|$. We also choose two random hash functions called G and H, that is:

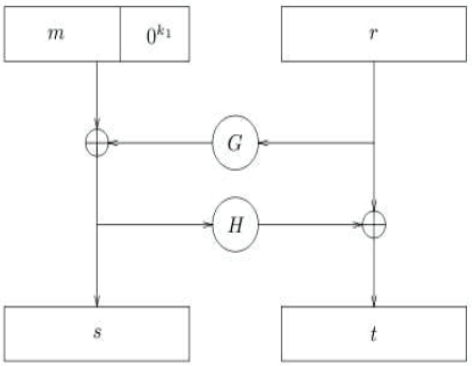
$$G : \{0,1\}^{k_0} \rightarrow \{0,1\}^{n-k_0}$$
$$H : \{0,1\}^{n-k_0} \rightarrow \{0,1\}^{k_0}$$

After this setup, we run OAEP algorithm to pad the message M so that we can generate stronger signature.

OAEP Algorithm
Input: M, k_0 and k_1, where $k_0, k_1 \in \mathbb{N}$
Output: M_{padded}

1: $r := \{0,1\}^{k_0}$
2: $G(r) := r \rightarrow \{0,1\}^{n-k_0}$
3: $s := G(r) \oplus (m||\{0,1\}^{k_1})$
4: $H(s) := s \rightarrow \{0,1\}^{k_0}$
5: $t := H(s) \oplus r$
6: $M_{padded} := (s||t)$
7: return M_{padded}

Figure 2: OAEP Scheme



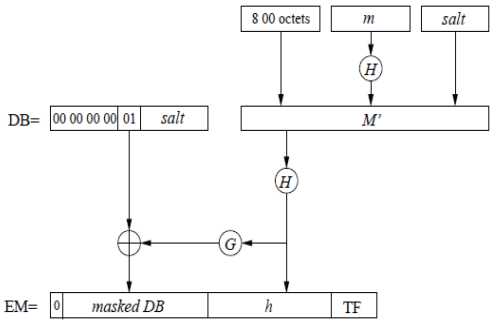
Using this padded message, M_{padded}, we use RSA signing equation $s = (M_{padded})^e \ mod \ n$ to create a signature.

B. Probabilistic Signature Scheme

PSS, Probabilistic Signature Scheme, is also another scheme that is intended to be both efficient and provably secure [9]. Unlike standard RSA and OAEP algorithms,

we can create a digital signature using PSS algorithm's signature equation. Same as OAEP, PSS also pad the message using two random padding variable. However, unlike OAEP, PSS algorithm returns EM value instead, but by using PSS signature equation, $s = EM^d \ mod \ n$ equation, we can find a signature. At first, we create a M' by concatenating fixed padding, $padding_1$, $H(M)$ and $salt$, which is a randomly generated octet string [10]. Then we pass this value M' into hash function again, $H(M')$ and then put into G function to make same length as DB, which is concatenation of another padding, $padding_2$ string and $salt$ value that we got. After we pass $H(M')$ into G function, we XOR the value with DB and let it equal to $maskedDB$ where $maskedDB = G(H(M')) \oplus DB$. At the end, we make EM by concatenating 0, $maskedDB$, $H(M')$, and another padding called TD, trailer field, which consists of single octet [10]. Since we know that $s = EM^d \ mod \ n$ which is equal to $s = (EM)M^{d-1} \ mod \ n$, we can finally generate a s from EM value that we got.

Figure 3: PSS Scheme



The advantages of both OAEP and PSS scheme are that at first, we pad message M with random number(s) and it makes the message more vary and hard to guess for attackers. Also by using hash functions, which is one-way function, it makes message more difficult to invert or in other words, it makes the message becomes more random or probabilistic.

VI. Conclusion

Using a digital signature scheme, we can authenticate the sender's identity and assure the recipient. Despite of the fact that RSA is an old algorithm among other public-key techniques, by using schemes that are mentioned in PKCS #1 [11], we can make it much stronger by randomizing through padding. For example in RSA-PSS, we use hash function almost three times and because of the fact that hash function is infeasible to solve in the other way, it makes message hard to break. But although using these schemes in PKCS #1 would make signature stronger, there are some disadvantages we can find. First, even though we use PKCS #1 schemes to make the signature much stronger, because our signature is going to be in 160 bits, it is still somewhat weak compare with encrypting the whole message. Another disadvantage is that because RSA is much slower than other symmetric cryptosystems in general [12], doing this whole digital signature process is going to be really slow and perhaps the next or future work for the RSA digital signature is going to be, rather than making the algorithm more secure, make the algorithm more efficiently in fast speed.

References

[1] J. G. Savard, "Digital signature based on diffie-hellman," `http://www.quadibloc.com/crypto/pk050302.htm`, June 2010.

[2] S. Burnett and S. Paine, *RSA Security's Official Guide to Cryptography*, McGraw-Hill, 2001.

[3] E. Abdel-Azeem, R. Seireg, and S. I. Shaheen, "Cryptographic security evaluation of md4 hash function," in *Radio Science Conference, 1996. NRSC '96., Thirteenth National*, mar 1996, pp. 345 –354.

[4] J. A. Buchmann, *Introduction to Cryptography*, Springer, 2004.

[5] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[6] M. C. Chu-Carroll, "Cryptographic padding in RSA," `http://scienceblogs.com/goodmath/2009/01/cryptographic_padding_in_rsa.php`, June 2010.

[7] "Optimal asymmetric encryption padding," `http://en.wikipedia.org/wiki/Optimal_Asymmetric_Encryption_Padding`, June 2010.

[8] S. AlZaabi, S. Baniabdalsalam, and M. Baniabdalsalam, "Symmetric encryption," 2008.

[9] A. Menezes, "Evaluation of security level of cryptography: RSA-OAEP, RSA-PSS, RSA signature," Tech. Rep., University of Waterloo, 2001.

[10] B. Kaliski, "Raising the standard for RSA signatures: RSA-PSS," Tech. Rep.

[11] "PKCS #1 v2.1: RSA cryptography standard," Tech. Rep., RSA Laboratories, 2002.

[12] "RSA," `http://en.wikipedia.org/wiki/RSA#Speed`, June 2010.