Modified Chinese Remainder Theorem and Its Application to Proxy Signatures

Chuan-Kun Wu and Vijay Varadharajan *

Abstract.
Chinese Remainder Theorem has been used for hundreds of years and has been applied to many domains such as integers and polynomials. An assumption made is that the component moduli are pairwise co-prime. In this paper, first we remove this assumption; then we give an algorithm to find whether a given system of congruent equations has a solution, and if so, how to find the solution in an efficient manner. Further we apply the modified Chinese Remainder Theorem to design proxy signatures.

Key words: *RSA public key system, Chinese remainder theorem, proxy signature*

1 Introduction

The original version of Chinese Remainder Theorem is believed to be first observed by a Chinese mathematician in the first century (between 280 and 473 A.D.). It has become such an important theorem that every number theory book has a description of it. The Chinese Remainder Theorem for integers considers a system of congruent equations

$$\begin{cases} x \bmod n_1 = a_1 \\ x \bmod n_2 = a_2 \\ \ldots\ldots \\ x \bmod n_k = a_k \end{cases} \tag{1}$$

where the moduli $n_i, i = 1, 2, ..., k$ are pairwise co-prime numbers, i.e., $\mathtt{gcd}(n_i, n_j) = 1$ for every pair $i \neq j$, where $\mathtt{gcd}()$ means the greatest common divisor. For every collection of $a_i < n_i,\ i = 1, 2, ..., k$, there exists an unique x in the sense of modulo $N = \prod_{i=1}^{k} n_i$ such that x mod $n_i = a_i$ holds for all $i = 1, 2, ..., k$, i.e., x is a solution of equation (1).

Note that in the Chinese Remainder Theorem described above, the condition $\mathtt{gcd}(n_i, n_j) = 1$ for every pair $i \neq j$ guarantees the existence of a solution x. However, if we consider a particular collection of $a_i < n_i,\ i = 1, 2, ..., k$, rather than an arbitrary collection (this is the case that often arises in practical applications), then the condition for Chinese remainder theorem is sufficient, but not necessary. A natural question is, what happens if the condition that $n_i,\ i = 1, 2, ..., k$ are pairwise co-prime is removed? It is known that there may not exist a solution for equation (1). An extreme example is that there does not exist a solution for

$$\begin{cases} x \bmod n = 1, \\ x \bmod n = 2. \end{cases}$$

Certainly there are cases where there does exist a solution. The question is under what condition[1] a solution of equation (1) exists? If a solution exists, how to find it? As far as the authors know, these problems have not been solved yet, though there are a number of different forms of the Chinese Remainder Theorem such as the one for integral polynomials. In this paper we study the generation of Chinese Remainder Theorem for integers, called the Modified Chinese Remainder Theorem. Using this, we then consider more precisely the relationship between private and public keys in the well-known RSA [6] system. Then we apply the Modified Chinese Remainder Theorem to the design of proxy signature schemes.

2 Modified Chinese Remainder Theorem

We are now considering congruent equation (1) in the general case, i.e. where the condition that the moduli are pairwise co-prime is removed. Assume that there is a solution for x to the equation. Then we can write $x - a_i = k_i n_i$ for some integer k_i. Denoting $a_i' = a_i - x$, we have $\mathtt{gcd}(a_i', a_j') = \mathtt{gcd}(k_i n_i, k_j n_j)$. So we have

Lemma 1 *A necessary condition for equation (1) to have a solution is that there exists an integer α such that for every pair (i, j) with $i \neq j$, $\mathtt{gcd}(n_i, n_j)|\mathtt{gcd}(a_i - \alpha, a_j - \alpha)$, where $A|B$ means that A is a factor of B.*

Now we examine whether the necessary condition in lemma 1 is a sufficient condition as well. This is equivalent to proving that, if

$$\mathtt{gcd}(n_i, n_j)|\mathtt{gcd}(a_i, a_j) \tag{2}$$

for every $i \neq j$, then there must exist a solution for equation (1). Note that $a_i - \alpha$ may be negative; we only consider the case where the remainders are positive. How can this problem be overcome? It is trivial to prove that $\mathtt{gcd}(n_i, n_j)|\mathtt{gcd}(a_i, a_j)$ if and only if $\mathtt{gcd}(n_i, n_j)|\mathtt{gcd}(a_i \bmod n_i, a_j \bmod n_j)$. So, if $a_i' = a_i - \alpha$ is negative, just replace a_i' by $a_i' \bmod n_i$.

*Authors are with School of Computing and Information Technology, University of Western Sydney (Nepean), PO Box 10, Kingswood, NSW 2747, Australia. {c.wu, v.varadharajan}@uws.edu.au

[1] Here we are interested in how the properties of a_i rely on the properties of n_i and vice versa.

Let $L_i = \texttt{lcm}(n_1, n_2, ..., n_i)$ for $i = 1, 2, ...k$, where $\texttt{lcm}()$ means the least common multiple of the numbers inside the parentheses. We also let $L = L_k$ to be the least multiple of all the moduli. First we give

Lemma 2 (Extended Euclid Algorithm) *Let n and m be two integers and $d = \texttt{gcd}(m, n)$. Then there must exist integers a and b such that $am + bn = d$. Moreover, there is an efficient algorithm to find a, b and d given m and n.*

Given integers n_i and a_i, $i = 1, 2, ..., k$, such that $\texttt{gcd}(n_i, n_j)|\texttt{gcd}(a_i, a_j)$ for every pair $i \neq j$, we determine whether there exists a solution for equation (1). First we give the algorithm and then prove that the algorithm works.

Algorithm 1

- *Set $x_1 = a_1$.*
- *If x_t ($1 \leq t \leq k-1$) is given. Let $d = \texttt{gcd}(L_t, n_{t+1})$. Using lemma 2 to find u and v such that*

$$uL_t + vn_{t+1} = d. \tag{3}$$

Set

$$x_{t+1} = (a_{t+1}L_t u + x_t n_{t+1} v)/d \bmod L_{t+1}. \tag{4}$$

- *Output sequence x_t, $t = 1, 2, ..., k$.*

Before we prove that Algorithm 1 works, the following two lemmas are needed.

Lemma 3 *Suppose $\texttt{gcd}(a, b)|\texttt{gcd}(c, d)$. Then for any number z, $\texttt{gcd}(az, b)|\texttt{gcd}(cz, d)$ holds if and only if $\texttt{gcd}(az, b)|d$ holds.*

Proof: Necessity is obvious. So we only need to prove the sufficiency. Let $e = \texttt{gcd}(az, b)$, $e_1 = \texttt{gcd}(e, a)$, $e_2 = e/e_1$. Then we must have $e_2|z$. Since $e_1|\texttt{gcd}(a, b)$ and $\texttt{gcd}(a, b)|\texttt{gcd}(c, d)$, we have $e_1|c$. Hence $e_1e_2|cz$. By the assumption that $e|d$ we have $e|\texttt{gcd}(az, b)$ and the conclusion of lemma 3 follows. □

Lemma 4 *For arbitrary numbers a, b, c, we have*[2] $\texttt{gcd}(\texttt{lcm}(a, b), c) = \texttt{lcm}(\texttt{gcd}(a, c), \texttt{gcd}(b, c))$.

Now we prove that Algorithm 1 works out a solution for equation (1) if condition (2) is satisfied.

Lemma 5 *The sequence produced by Algorithm 1 satisfies the following properties:*

$$x_t < L_t. \tag{p1}$$
$$x_t \bmod L_{t-1} = x_{t-1} \text{ if } t > 1. \tag{p2}$$
$$x_t \bmod n_t = a_t. \tag{p3}$$
$$\texttt{gcd}(L_t, n_i)|\texttt{gcd}(x_t, a_i) \text{ for all } i > t \text{ if } t < k. \tag{p4}$$

Proof: It can be proved deductively.

It is obvious that x_1 satisfies properties $p1$, $p3$ and $p4$ (property $p4$ comes from assumption (2). From equations (3) and (4) it is also known that x_2 satisfies properties $p1$, $p2$ and $p3$. It can also be proved that x_2 satisfies property $p4$ as well. Similar arguments can be made for the general case of x_{t+1}.

Now suppose we have sequence x_j for $j \leq t$ which satisfy all the properties $p1, p2, p3$ and $p4$ (except x_1 where property $p2$ does not make sense). The conclusion of the theorem is proved if we can prove that x_{t+1} from equation (4) satisfies all the following properties:

(1) $x_{t+1} < L_{t+1}$.
(2) $x_{t+1} \bmod L_t = x_t$.
(3) $x_{t+1} \bmod n_{t+1} = a_{t+1}$.
(4) $\texttt{gcd}(L_{t+1}, n_i)|\texttt{gcd}(x_{t+1}, a_i)$ for all $i > t+1$.

It is apparent that property **(1)** is satisfied. By the assumption, x_t satisfies property $p4$, so $d = \texttt{gcd}(L_t, n_{t+1})$ is a factor of $\texttt{gcd}(x_t, a_i)$. As a particular case we have $d|\texttt{gcd}(x_t, a_{t+1})$. Therefore by equation (3), $x_{t+1} \bmod L_t = \frac{x_t}{d}n_{t+1}v \bmod L_t = x_t$. This proves property **(2)**. Similar to the proof of property **(2)** we can prove property **(3)**, i.e., $x_{t+1} \bmod n_{t+1} = a_{t+1}$. Now we prove property **(4)**. Note by equation (3) that $\texttt{gcd}(L_{t+1}, n_i) = \texttt{gcd}(\texttt{lcm}(L_t, n_{t+1}), n_i) = \texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)$, property **(4)** is equivalent to

$$\texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|\texttt{gcd}(\frac{a_{t+1}L_t u}{d} + \frac{x_t n_{t+1} v}{d}, a_i).$$

This is true if we can prove

$$\begin{cases} \texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|\texttt{gcd}(\frac{a_{t+1}L_t u}{d}, a_i), \\ \texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|\texttt{gcd}(\frac{x_t n_{t+1} v}{d}, a_i). \end{cases}$$

Furthermore, this is true if we can prove

$$\begin{cases} \texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|\texttt{gcd}(\frac{a_{t+1}L_t}{d}, a_i), \\ \texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|\texttt{gcd}(\frac{x_t n_{t+1}}{d}, a_i). \end{cases}$$

For the property $\texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|\texttt{gcd}(\frac{a_{t+1}L_t}{d}, a_i)$, since $\texttt{gcd}(n_{t+1}, n_i)|\texttt{gcd}(a_{t+1}, a_i)$, by lemma 3 it is true if we can prove $\texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|a_i$. For the property $\texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|\texttt{gcd}(\frac{x_t n_{t+1}}{d}, a_i)$, since $\texttt{gcd}(L_t, n_i)|\texttt{gcd}(x_t, a_i)$ (property $p4$ of x_t), by lemma 3 it is true if we can prove $\texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|a_i$. Thus property **(4)** is true if we can prove $\texttt{gcd}(\frac{L_t n_{t+1}}{d}, n_i)|a_i$, or equivalently $\texttt{gcd}(L_{t+1}, n_i)|a_i$.

From the property $p4$ of x_t, $\texttt{gcd}(L_t, n_i)|\texttt{gcd}(x_t, a_i)$ implies that $\texttt{gcd}(L_t, n_i)|a_i$; from the initial assumption of Algorithm 1, $\texttt{gcd}(n_{t+1}, n_i)|\texttt{gcd}(a_{t+1}, a_i)$ implies that $\texttt{gcd}(n_{t+1}, n_i)|a_i$. So $\texttt{lcm}(\texttt{gcd}(L_t, n_i), \texttt{gcd}(n_{t+1}, n_i))|a_i$. Note that by lemma 4 we have

$$\begin{aligned} \texttt{gcd}(L_{t+1}, n_i) &= \texttt{gcd}(\texttt{lcm}(L_t, n_{t+1}), n_i) \\ &= \texttt{lcm}(\texttt{gcd}(L_t, n_i), \texttt{gcd}(n_{t+1}, n_i)), \end{aligned}$$

this proves $\texttt{gcd}(L_{t+1}, n_i)|a_i$ and consequently x_{t+1} satisfies all the designated properties.

From the proof described above we can conclude the correctness of lemma 5. □

Note: In the proof of lemma 5, the expression of x_{t+1} in equation (4) is used but the modulus L_{t+1} is not considered. This does not affect the correctness of the conclusion, as it is obvious that $\texttt{gcd}(L_{t+1}, n_i)|\texttt{gcd}(x_{t+1}, a_i)$ if and only if $\texttt{gcd}(L_{t+1}, n_i)|\texttt{gcd}(x_{t+1} \bmod L_{t+1}, a_i)$.

[2] There is another complimentary associative property of $gcd()$ and $lcm()$ which says: $\texttt{lcm}(\texttt{gcd}(a, b), c) = \texttt{gcd}(\texttt{lcm}(a, c), \texttt{lcm}(b, c))$. This property is not used in this paper.

Lemma 6 *If the property in equation (2) is satisfied, then equation (1) has a unique solution which can be generated using Algorithm 1.*

Proof: By lemma 5 we know that Algorithm 1 produces a sequence $\{x_i\}$ satisfying properties $p1, p2, p3$ and $p4$. Now we prove that if $x_t \bmod n_i = a_i$ for all $i \leq t$, then $x_{t+1} \bmod n_i = a_i$ holds for all $i \leq t+1$. Since $x_{t+1} \bmod L_t = x_t$, and for all $i \leq t$, $n_i | L_t$, so we have $x_{t+1} \bmod n_i = (x_{t+1} \bmod L_t) \bmod n_i = x_t \bmod n_i = a_i$ for all $i \leq t$. In addition the congruence $x_{t+1} \bmod n_{t+1} = a_{t+1}$ is known to be true. Thus the final constructed x_k is a solution of the equation (1). Now we prove the uniqueness. Let us assume the contrary that there are two solutions x and y. Since they satisfy $x \bmod n_i = a_i$ and $y \bmod n_i = a_i$, we have $x - y \bmod n_i = 0$, or equivalently $n_i|(x-y)$. This implies that $L_k|(x-y)$. But $x < L_k$ and $y < L_k$, so we must have $x = y$ and the uniqueness of the solution is proved. □

In the general case, given equation (1), if there exists α such that for every pair (i, j) with $i \neq j$, $\texttt{gcd}(n_i, n_j)|\texttt{gcd}(a_i - \alpha, a_j - \alpha)$, then by finding a solution for congruence $x' \bmod n_i = a_i - \alpha$, the designated solution for the original equation is $x = x' + \alpha$. Furthermore, by combining lemma 1 and lemma 6, we have

Theorem 1 *There exists a solution for equation (1) if and only if there exists an integer α such that $\texttt{gcd}(n_i, n_j)|\texttt{gcd}(a_i - \alpha, a_j - \alpha)$ holds for every $i \neq j$. Furthermore, the solution is unique in the modulus $L = \texttt{lcm}(n_1, n_2, ..., n_k)$.*

3 Working out a solution

Given equation (1), if it has the property of equation (2), then Algorithm 1 can work out the solution. In the general case, if we can find α such that $\texttt{gcd}(n_i, n_j)|\texttt{gcd}(a_i - \alpha, a_j - \alpha)$ for every $i \neq j$, then by replacing a_i with $a_i' = a_i - \alpha$, we can find a solution x'; the solution for the original equation is $x = x' + \alpha$. However, even if such an α exists, sometimes it is hard to find. For example, the minimum positive α satisfying the above described property for equation

$$\begin{cases} x \bmod 10 = 6 \\ x \bmod 6 = 4 \\ x \bmod 15 = 1 \end{cases}$$

is the solution $x = 16$ itself. This means that to find such an integer α can be as hard as to find the solution itself for the equation. In general, α could be an arbitrary number between 0 and $L = \texttt{lcm}(n_1, n_2, ..., n_k)$. By testing every possible number in $[0, L-1]$ it can be determined whether such an α exists. However, when L is a very large number, this method is very inefficient. So we need to find a more efficient way to find the solution when it exists, or to report the non-existence of a solution.

We first examine the system of two modulo equations

$$\begin{cases} x \bmod n_1 = a_1, \\ x \bmod n_2 = a_2. \end{cases} \tag{5}$$

By theorem 1 we know that equation (5) has a solution if and only if there exists α such that $d = \texttt{gcd}(n_1, n_2)|\texttt{gcd}(a_1 - \alpha, a_2 - \alpha)$.

Lemma 7 *In equation (5), there exists an integer α such that $d|\texttt{gcd}(a - \alpha, b - \alpha)$ if and only if $d|(a - b)$.*

By lemma 7, for the case of equation (5) where there are only two modulo equations, it is not necessary to test all possible values of α. Instead, by letting $\alpha = a_1$ (or equivalently $\alpha = a_2$) the existence of the solution of equation (5) can be checked. This greatly reduces the complexity compared to testing α in the general case. If a solution exists, then we solve equation

$$\begin{cases} x' \bmod n_1 = a_1 - a_2, \\ x' \bmod n_2 = 0. \end{cases} \tag{6}$$

Apparently the existence of x for equation (5) is sufficient for the existence of x' for equation (6), since we can set $x = x' + a_2 \bmod \texttt{lcm}(n_1, n_2)$. It is known that x' must be a multiple of d. Let $x'' = x'/d$, $n_1' = n_1/d$, $n_2' = n_2/d$, and $a = (a_1 - a_2)/d$. It is known that $gcd(n_1', n_2') = 1$. By the traditional Chinese remainder theorem, it is easy to find a solution for

$$\begin{cases} x'' \bmod n_1' = a, \\ x'' \bmod n_2' = 0. \end{cases} \tag{7}$$

Consequently we have $x' = dx''$ and $x = x' + a_2$. So we have

Lemma 8 *Equation (5) has a solution x if and only if $\texttt{gcd}(n_1, n_2)|(a_1 - a_2)$. If it does, then $x = dx'' + a_2$ is a solution and is unique in the sense of modulo $\texttt{lcm}(n_1, n_2)$, where x'' is a solution of equation (7).*

Based on lemma 8, we can develop a method for finding a solution of equation (1) in the general case.

Algorithm 2

Input *Sequences $\{n_i\}$ and $\{a_i\}$ with $a_i < n_i$, $i = 1, 2, ..., k$.*

Output *A solution x for the equation (1), or to report the non-existence of a solution of equation (1).*

- *Solve equation (5) for x_1. If no solution exists, then report the non-existence of solution of equation (1) and stop.*
- *For $t = 1$ to $k-1$, given x_t, if $\texttt{gcd}(L_t, n_{t+1}) \not| (x_t - a_{t+1})$, then report the non-existence of solutions to equation (1) and stop. Else solve the equation*

  $$\begin{cases} x_{t+1} \bmod L_t = x_t, \\ x_{t+1} \bmod n_{t+1} = a_{t+1}. \end{cases}$$

  - *Let $d = \texttt{gcd}(L_t, n_{t+1})$.*
  - *Set $m = L_t/d$, $n = n_{t+1}/d$, $a = (x_t - a_{t+1})/d$.*
  - *Compute $n^{-1} \bmod m$ (refer lemma 2), let $x = ann^{-1}$.*
  - *Then $x_{t+1} = dx + a_{t+1} \bmod \texttt{lcm}(L_t, n_{t+1})$.*

- *Output x_k which is a solution of equation (1).*

It can be seen that Algorithm 2 is similar to Algorithm 1 except that a transformation is made and the traditional Chinese remainder theorem is used in finding a solution of equation systems with two modulo equations.

3.1 Computational Complexity

We now discuss briefly the computational complexity of Algorithm 2 compared with that of traditional Chinese remainder theorem when k pairs of moduli and remainders (n_i, r_i) are given. When the moduli are pairwise co-prime, to compute a solution, the traditional Chinese remainder theorem makes use of k times of the Extended Euclid algorithm (lemma 2). A further optimization can reduce the use of Extended Euclid algorithm [3] to $k-1$ times. In Algorithm 2, from the first pair of modulus and remainder, every additional pair needs to use the Extended Euclid Algorithm for two times, i.e., the algorithm makes use of $2k-2$ times of the Extended Euclid algorithm. In solving every system of equations with only two modulo equations, the first use of the Extended Euclid Algorithm is to check whether a solution exists, and the second use is, like in traditional Chinese remainder theorem, to find the solution. Apart from the Extended Euclid Algorithm (which can be implemented very efficiently[3]), other computations are normal operations, namely modulo addition, subtraction, multiplication and division.

4 Selection of RSA private keys

A RSA public key system works like this: Let $n = pq$ be the product of two distinct large primes, e and d be two random numbers satisfying $ed \equiv 1 \pmod{\phi(n)}$, where $\phi(n) = (p-1)(q-1)$ is the Euler function on n. Then the pair (n, e) is set as the public key and d is kept as the private key. Given a message $m < n$, encryption is done as $c = m^e \bmod n$, and the corresponding decryption algorithm is $m = c^d \bmod n$. The mechanism of RSA is based on Euler theorem which says: For any co-prime numbers a and n, $a^{\phi(n)} \bmod n = 1$ is always true. The security of RSA system is based on the assumption of intractability of factoring large integers.

In general, it is assumed that when a RSA key is to be changed, both the public and the private keys have to be changed accordingly as they need to satisfy the relation $ed \bmod \phi(n) = 1$. In fact, this is not necessary. In practical implementations, another relationship is used (actually replace $\phi(n)$ by another function $\lambda(n)$) which is shown below.

For an arbitrary positive integer n, denote by $Z_n = \{0, 1, ..., n-1\}$ to be the remainder ring modulo n, and Z_n^* the largest multiplicative group within Z_n. It is known that the order of Z_n^*, or the number of elements in Z_n^*, is $\phi(n)$. For an element $\alpha \in Z_n^*$, define its order to be the smallest positive integer k such that $\alpha^k \bmod n = 1$, and denoted by $ord_n(\alpha) = k$. Define $\lambda(n)$ to be the maximum value of $ord_n(\alpha)$ for all $\alpha \in Z_n^*$. As Z_n^* exists only when $n \geq 2$, the definition of $\lambda(n)$ makes sense only when $n \geq 2$. We have

Lemma 9 ([7]) *The value of function $\lambda(n)$ can be computed as follows:*

- $\lambda(2) = 1$, $\lambda(4) = 2$, *and* $\lambda(2^k) = 2^{k-2}$ *for* $k \geq 3$.
- *If p is an odd prime, then $\lambda(p^k) = p^{k-1}(p-1) = \phi(p^k)$ for any positive integer k.*
- *If $n = n_1 n_2 \cdots n_k$, where n_i are pairwise co-prime, then* $\lambda(n) = \mathtt{lcm}(\lambda(n_1), \lambda(n_2), \cdots, \lambda(n_k))$.

Note that for any positive integer n, $\lambda(n) | \phi(n)$. So we also have

Theorem 2 *Given RSA public key (n, e), then d can be a private key which works for all messages*[4] *if and only if*

$$ed \equiv 1 \pmod{\lambda(n)}. \tag{8}$$

Proof: Trivial. □

As mentioned above, in the implementation of RSA, equation (8) in theorem 2 is used rather than the relationship that $ed \equiv 1 \pmod{\phi(n)}$. However, this does not mean that the private key corresponding to a given public key is unique. In fact, if e is a public key and d is a private key, then for an arbitrary integer k, $d + k\lambda(n)$ can be another private key as well. This property also applies to public key. At this stage, no benefits from this multiple choice of private/public keys can be seen. This will be reflected in the applications in the next section.

5 A proxy signature scheme

In this section we show one of the applications of the Modified Chinese Remainder Theorem. This is expected to show the significance of the Modified Chinese Remainder Theorem and does not mean that this is the only application of Modified Chinese Remainder Theorem.

Proxy signature is proposed in [4]. The purpose for proxy signature is to enable proxies to sign documents on behalf of a user (it is useful when the user is for some reason not available and the documents need to be processed in which case they have to be authorized by another acting on behalf of the original user). A secure proxy signature scheme should have the following properties:

Verifiability. Validity of proxy signatures can be verified publicly.

Unforgeability. Only the original signer and the designated proxy can create a valid proxy signature.

Identifiability. In certain circumstances, proxy signatures should be identified from the normal signatures created by the original user.

[3] Extensive analysis on the complexity of Extended Euclid algorithm can be found in [3]. It is known that even in the worst case, the computational complexity of the Extended Euclid algorithm is a polynomial of the size of the input.

[4] This means that when RSA is used for encryption, then all the ciphertexts encrypted using public key e can be correctly decrypted using d; while when RSA is used for creating digital signatures, for every message $m < n$, $m^d \bmod n$ is a valid signature.

In [2], different types of proxy signatures have been reviewed. Basically there are three fundamental types of proxy signatures, namely full delegation, partial delegation, and delegation by warrant. The *full delegation* as the name implies allows complete delegation of all privileges to another user. The *delegation by warrant* needs a signature of the original user on the warrant and hence verification of a proxy takes longer time, as it is actually a verification of two signatures. In a *partial delegation* scheme, the original user generates a different pair of private/public key to the proxy delegate, and declares that the public key is authorized for proxy signature verification. In [2] two new types of proxy signatures have been introduced. In this paper we introduce a RSA-based proxy signature scheme which can be classified as partial delegation. It should be noticed that, to the best of the authors' knowledge, this is the first proxy signature scheme based on RSA system.

Set up

Consider a RSA signature system where a user Alice has a public key (n, e) authorised and published. Alice has a private key d satisfying that $ed \bmod \lambda(n) = 1$. Suppose that Alice wants to have k proxies which can sign less important documents on her behalf. She does the following computation.

- Let $\lambda_i = p_i\lambda(n)$, where p_i are chosen such that they satisfy

$$p_i | p_{i+1} \text{ and } \mathtt{gcd}(p_i, e) = 1. \tag{9}$$

- Compute d_i such that

$$\begin{cases} ed_i \bmod \lambda_i = 1, \\ ed_i \bmod \lambda_j \neq 1 \text{ for all } j > i. \end{cases} \tag{10}$$

  This enables $d_i \neq d_j$ for all $i \neq j$.

- Send d_i securely to the proxy $\mathcal{P}_i$.

- Choose g such that $ord_n(g) = \lambda(n)$ and publish g with her public key. Then authorise (by signing it) and publish $G_i = g^{d_i} \bmod \lambda(n)$ associated with the identity of proxy $\mathcal{P}_i$ (so that the proxies cannot modify the published data).

Let us consider whether the equation (10) can be satisfied. Let $x \bmod \lambda_i = 1$ and $x \bmod \lambda_j = r_j\lambda(n) + 1$ for all $j > i$, where r_j is a random number and $r_j\lambda(n) + 1 < \lambda_j$. Then using the Modified Chinese Remainder Theorem (theorem 1), there must be a solution for x. Letting $ed_i \bmod \lambda_i = x$, we have $d_i = xe^{-1} \bmod \lambda_i$.

Signing and verification

When proxy $\mathcal{P}_i$ is requested to sign a message m, he creates the signature as $s = m^{d_i} \bmod n$. Verification of the validity of the proxy signature is as simple as verifying a normal RSA signature. The signature is accepted if $s^e \bmod n = 1$. This verification applies to signatures by all the proxies and those created by the original user as well. However, the signature verification cannot tell whether a signature is created by a proxy or the original signer.

Signature confirmation

The receiver of a signature may (sometimes) need to know which proxy signed the message. It is assumed that in producing the signature, the receiver is informed in plain text which proxy created the signature. The next issue is to confirm the correctness of the proxy's identity. We make use of G_i, a reference signature of $\mathcal{P}_i$ on g, with which the identity of $\mathcal{P}_i$ is associated. This can be done using a one-round interactive zero-knowledge proof (both the idea and the proof are similar to those in [1]). Let $\mathcal{P}$ denote the prover (proxy $\mathcal{P}_i$) and $\mathcal{V}$ denote the verifier (the receiver). The following protocol confirms to $\mathcal{V}$ that a signature s is made by proxy $\mathcal{P}_i$.

Protocol C: proxy ID confirmation

$\mathcal{V} \longrightarrow \mathcal{P}$:	Proxy ID confirmation request.
$\mathcal{P} \longrightarrow \mathcal{V}$:	$\mathcal{P}_i'$s identity.
$\mathcal{V}$:	Choose random integers r_1, r_2.
$\mathcal{V} \longrightarrow \mathcal{P}$:	$Q := m^{r_1}g^{r_2} \bmod n$.
$\mathcal{P} \longrightarrow \mathcal{V}$:	$A := Q^{d_i} \bmod n$.
$\mathcal{V}$:	Verify whether $A == s^{r_1}G_i^{r_2} \bmod n$.

Protocol **C** is similar to the confirmation protocol of undeniable signature proposed in [1]. Its security can be proved in a similar fashion.

Properties

The properties of this proxy signature scheme include the following.

Verifiability It is obvious that the verification of proxy signatures is identical to the verification of normal RSA signatures which can be done publicly.

Identifiability The confirmation protocol (protocol **C**) is used to confirm the receiver the identity of the proxy.

Unforgeability No proxy can forge any other proxy's signatures. This is because every proxy has an unique reference signature G_i which is authorised by the original user.

Soundness Conspiring proxies do not get more information. From equation (9) and applying the Modified Chinese Remainder Theorem to equation (10), it is known that any two or more conspiring proxies can get $r\lambda_i$, where i is the largest index of these $\mathcal{P}_i$ and r is an unknown (random) number. From this one may be able to find a number d' which satisfies $ed' \bmod \lambda_i = 1$ and forge signatures. However as there is no reference signature on it, it can easily be identified using the confirmation protocol (protocol C).

Modifying the original signature

When a signature is to be verified, it would be desirable that verification itself can tell whether the signature is created by the original user or a proxy delegate. However the verification above is identical to that of normal RSA signatures. In order to tell the difference between an original signature and a proxy signature, we need to modify the normal RSA signature verification.

Assume that we have a one way hash function $h(x)$ which maps arbitrary length message into a hashed value smaller than n. When a message m is signed using a private key d' (d' could be the private of the original user or of any of the proxies), the signature is produced as $s = h(m)^{d'} \bmod n$, and hence verification is modified accordingly as checking whether $h(m) == s^e \bmod n$ (below we will write $h(m)$ as $\overline{m}$). Let the private key d of the original user satisfy that (note that $\lambda(n)$ is an even number)

$$\begin{cases} ed \bmod \lambda(n)/2 = 1, \\ ed \bmod \lambda(n) \neq 1. \end{cases} \tag{11}$$

From the Modified Chinese Remainder Theorem, we can see that there is a solution. Then when the order of $\overline{m}$ in Z_n^* reaches the maximum value $\lambda(n)$, verification yields

$$s^e \bmod n = \overline{m}^{ed} \bmod n \neq \overline{m}. \tag{12}$$

However, it also satisfies that

$$(s^e)^2 \bmod n = \overline{m}^2 \bmod n. \tag{13}$$

If a signature satisfies equations (12) and (13), then it is treated as the original signature; otherwise it is a proxy signature and further proxy identification protocol may need to run. The problem now is how likely the order of $\overline{m}$ reaches the maximum value. We have the following

Lemma 10 *Let p be a prime, then there are $\phi(p-1)$ primitive elements in Z_p^*, i.e., there are $\phi(p-1)$ elements having the maximum order.*

Lemma 11 *Let p and q be different primes, $ord_p(x) = a$ and $ord_q(x) = b$. Then $ord_{pq}(x) = \mathtt{lcm}(a, b)$.*

Lemma 12 *Let $x \in Z_n^*$ be a random number. Then $Prob(ord_n(x) = \lambda(n)) = \frac{\phi(p-1)}{p} \cdot \frac{\phi(q-1)}{q})$.*

Proof:

$$\begin{array}{rl} & Prob(ord_n(x) = \lambda(n)) \\ = & 1 - Prob(ord_n(x) < \lambda(n)) \\ \geq & 1 - Prob(ord_p(x) \leq \lambda(p))Prob(ord_q(x) \leq \lambda(q)) \\ & -Prob(ord_p(x) = \lambda(p))Prob(ord_q(x) = \lambda(q)) \\ = & 1 - (1 - \frac{\phi(p-1)}{p})(1 - \frac{\phi(q-1)}{q}) + \frac{\phi(p-1)}{p} \cdot \frac{\phi(q-1)}{q} \\ = & \frac{\phi(p-1)}{p} + \frac{\phi(q-1)}{q} \end{array}$$ $\square$

Lemma 12 shows that when $\phi(p-1)$ and $\phi(q-1)$ are fairly large compared with p and q respectively, for a randomly chosen message m, there is a high probability that the order of $\overline{m}$ reaches the maximum value in Z_n^*. Note that this does not guarantee that every signature created by the original user can be recognized by the verifier. So it is required that $\overline{m}$ satisfies equations (12) and (13). For a text message, this is not a problem, as the user can always modify the message by means of say changing a semicolon into a full stop, or modify the literal expression if necessary, so that different versions of a same message are produced. It is assumed that the hashed value behaves as a random number. So by modifying the format of a message one can get a version for which the hashed value $\overline{m}$ satisfies equations (12) and (13). However, although this enables the original signature to be recognized in signature verification, it is not sufficient to protect security, because this would lead to the factorization of n as $n|(s^{2e} - \overline{m}^2)$ (see equation (13)). The only case where the factorization of n cannot be done is that $s = -\overline{m} \bmod n$. So the original can safely create a signature if and only if $\alpha = s^e \bmod n = n - \overline{m}$, i.e., $\overline{m}^{ed} \bmod n = -\overline{m} \bmod n$. Now the problem occurs when this equation holds. We have

Lemma 13 *Let $a \in Z_n$ and let $x = -a^2 \bmod n$. Then $x^{ed} \bmod n = -x \bmod n$.*

Proof:

$$\begin{array}{rl} & x^{ed} \bmod n \\ = & (-a^2)^{ed} \bmod n \\ = & -a^{2ed} \bmod n \text{ (because } e, d \text{ are all odd numbers)} \\ = & -a^2 \bmod n \text{ (equation (13)} \\ = & -x \bmod n. \end{array}$$ $\square$

It is known [5] that there are $\phi(n)/4$ quadratic residues modulo n. So the original user is expected to modify the message for 4 times before a satisfied hashed value is obtained. By doing this, an original signature can always be recognizable from proxy signature in the verification step.

References

[1] R.Gennaro, H.Krawczyk, and T.Rabin, "RSA-based undeniable signatures", *Advances in Cryptology — Crypto'97*, B.S. Kaliski Jr. (Ed.), LNCS 1294, Springer-Verlag, 1997, pp.132-149.

[2] S.Kim, S. Park and D. Won, "Proxy signatures, revisited", *Information and Communications Security, Proceedings*, LNCS 1334, Y.Han, T.Okamoto, S.Qing (Eds.) Springer-Verlag, 1997, pp.223-232.

[3] D.E. Knuth, *The Art of Computer Programming*, 3rd edition, Vol.2, Addison-Wesley, 1997.

[4] M. Mambo, K. Usuda, and E. Okamoto, "Proxy signatures". *Proceedings of the 1995 Symposium on Cryptography and Information Security (SCIS'95)*, Inuyama, Japan, 24-27 Jan. 1995, pp.B1.11-17.

[5] A.J.Menezes, P.C. van Oorschot and S.A.Vanstone, *Handbook of Applied Cryptography*, CRC Press 1997.

[6] R.L.Rivest, A.Shamir, and L.Adleman, " A method for obtaining digital signatures and public-key cryptosystems", *Comm. ACM*, Vol.21, No.2, 1978, pp.120-126.

[7] K.H.Rosen, *Elementary Number Theory and Its Applications,* Second ed., Addison-Wesley, 1988.

[8] C.K.Wu and V.Varadharajan, "Many-to-one Cryptographic Algorithms and Group Signatures (Extended Abstract)", *Proceedings of Australasian Computer Science Conference (ACSC'99)*, 18th - 21st January, New Zealand, 1999, Springer, pp.432-444.