Accelerating RSA Encryption Using Random Precalculations

Loránd Szőllősi, Tamás Marosits and Gábor Fehér
(Corresponding author: Loránd Szőllősi)

Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics
2. Magyar Tudósok krt., H-1117 Budapest, Hungary (Email: \{szollosi, marosits, feher\}@tmit.bme.hu)

(Received Nov. 13, 2008; revised Feb. 5, 2009; and accepted May 31, 2009)

Abstract

RSA encryption and digital signature algorithm is considered secure if keys are $1024 - 4096$ bits long. Since it requires modular exponentiation on numbers of this length, embedded systems need either a cryptographic co-processor or a fast CPU to calculate ciphertexts and signatures. In many applications, the sender is resource-scare, so optimization is necessary. In our paper we show a method for precalculations that accelerates the real-time performance of the sender in the expense of additional calculations at the receiver. When completed, the receiver gets an RSA-equivalent ciphertext for the encryption algorithm.

Keywords: Embedded systems, public key cryptography, RSA

1 Introduction

The RSA [5, 9, 11, 15] algorithm is one of the most common methods for public key encryption and signature creation. It is based on the intracability of factorization and discrete logarithm; the RSA function, when implemented, consists of one modular exponentiation. Despite its simplicity when formalized, RSA is actually quite challenging for developers of embedded systems, because it needs to work on $1024 - 4096$ bit long numbers to achieve adequate security level.

Since modular exponentiation is defined as repeated multiplications over a modular field, the most common method to speed up RSA is to use fast multiplication algorithms. In the range of RSA key lengths, Karatsuba [7] and Toom-Cook [13] algorithms leads to the fastest algorithms on embedded processors, while on devices capable of floating point calculations, Fast Fourier Transformation (FFT) [2] should also be considered. Number Theoretic Transformation (NTT) [12] is asymptotically faster than Karatsuba or Toom-Cook, and it does not require floating point calculations like FFT, but on current key lengths, it is somewhat slower in the embedded world [1, 16]. The theoretic lower bound of calculation requirement is given in [2], and NTT approximates it very closely, so further optimization of multiplication - albeit being theoretically interesting - is not expected to yield a speedup of multiple orders of magnitude.

Another approach is to speed up the method of exponentiation. While fast exponentiation [8] is the most common method to calculate RSA ciphertexts and signatures, it is possible to further optimize it using addition chains [4]. This yields a further speedup up in the RSA processing capabilities. Other implementations use extensive storage to speed up the calculation of addition chains (see [4] for a survey of these algorithms); these require a high amount of secure memory for signature calculation, and securely addressable memory for encryption.

Even using these algorithms, there remains a need for further speedup in the range of embedded devices, in order to eliminate the need of cryptographic co-processors. Devices used in sensor networks are typically ten times cheaper than cryptographic processors. Even if there remains a little room for optimization of the total execution time of RSA calculations, it might still be possible to do precalculations to increase the real-time performance of the encryption. Such algorithms are usually deterministic, and store a (redundant) set of base vectors or elements of the addition chain. Our main purpose has been to find a different, randomized precalculation model that has low resource requirements considering both memory and processing power.

2 Using Random Precalculation for Encryption with the Recommended Exponent

2.1 Notation

We will use the standard notation for RSA defined in [11]. Table 1 shows the variables used. Using this notation, the public key is $(n, e)$, the private key is $(n, d)$; a message can be encrypted as $c = m^e \pmod{n}$ and it can be decrypted as $m = c^d \pmod{n}$. 


Table 1: RSA notation

| $p$, $q$ | Large primes used to generate the key |
| $n = pq$ | Modulus |
| $\phi(n) = (p - 1)(q - 1)$ | Euler function of $n$ |
| $e$ | Public exponent |
| $d \equiv e^{-1} \pmod{\phi(n)}$ | Private exponent |
| $m$ | Message |
| $c$ | Ciphertext |

2.2 System Construction

2.2.1 Encryption

To come over the limit of deterministic addition chains, we first need to define a model in which we consider precalculations. We investigated a model where the ciphertext $c$ can be reproduced from a set of values $c_i$ (ciphertext parts) sent by the encoder using multiplication, exponentiation, and inverting over mod $n$. We investigated a model where the ciphertext $c$ can be reproduced from a set of values $c_i$ (ciphertext parts) sent by the encoder using multiplication, exponentiation, and inverting over mod $n$. This requires that $c_i$’s are composed of powers of the message $m$ and powers of some of the random numbers $r_i$. Therefore,

$$c_i = m^{k_{f(i,0)}} \prod_{j=1}^{r_i} r_i^{k_{f(i,j)}} \pmod{n},$$

where $k_i$ are constants and $f(\cdot, \cdot)$ is a mapping of $i,j$ indices to the indices of $k$. This scheme is secure only if the minimum power of $m$ that can be calculated in the form $\prod_i c_i^{k_i}$ without $r_i$ is $e$ (the public exponent). For high real-time performance, we try to reduce the number of ciphertext parts ($c_i$’s) that contain a nonzero power of $m$ (because these cannot be precalculated), and also we try to decrease these nonzero powers as much as possible. A simple special case of this format is where there are three ciphertext parts ($i = 1, 3$), and only $c_1$ and $c_2$ contain a power of the message. Thus, our formulation becomes:

$$c_1 = m^{k_1} r_1^{k_2}, \quad c_2 = m^{k_3} r_2^{k_4}, \quad c_3 = r_1^{k_5} r_2^{k_6}.$$

To find appropriate $k_i$ constants for the above equations, we first define the operator $\otimes$ over a vector and a matrix as:

$$(v \otimes M)_i = \prod_{j=1}^{\dim(v)} v_j^{M_{i,j}}.$$

This operator is used to simplify the calculation of exponents.

Using this operator, our formulation becomes (in vector notation):

$$v = \begin{bmatrix} m & r_1 & r_2 \end{bmatrix} \otimes \begin{bmatrix} k_1 & k_2 & 0 \\ k_3 & 0 & k_4 \\ 0 & k_5 & k_6 \end{bmatrix}.$$

All the possible combinations of powers of $c_i$ can be expressed as $v \otimes w$ where $w$ is a vector, because

$$\prod_i c_i^{k_i} = v \otimes K.$$

Since $\phi(n)$, the order of the group which the elements of $w$ are members of, is unknown to an attacker, inverse elements in the group of exponents are hard to calculate (note that an attacker might still be able to calculate the inverse over mod $n$ using the Extended Euclidean Algorithm). If the attacker knows $k_i$, he might try to calculate a power of $m$ lower than $e = 65537$ by a method similar to Gaussian Elimination, but without division by a constant. Our objective is to make it impossible to calculate $m^i$ without the knowledge of private parameters. To calculate a power of $m$ from $\xi$, one first needs to arrange the powers of $r_1$ and $r_2$ coefficients such that they are the same in two rows. Then one can invert one row (mod $n$), and multiply the two rows. Since the first row contains only $r_1$, while the second contains only $r_2$, the third row needs to be arranged such that multiplying the first two rows yields the same exponents of $r_{1,2}$ in both rows as coefficients. Thus, we multiply the first row by $k_3 k_4$, the second row by $k_5 k_2 k_3$ and the third row by $k_3 k_4$. This calculation assumes that $gcd(k_2, k_4) = 1$, $gcd(k_5, k_3) = 1$, $gcd(k_4, k_5) = 1$ (otherwise we could have divided the multiplication factors by their greatest common divisor), yielding the coefficients $r_1^{k_3 k_4}$ and $r_2^{k_5 k_2 k_3}$ in all three rows, while the powers of $m$ are $k_1 k_2 k_3, k_1 k_3 k_2$ and 0 respectively. So the matrix to calculate $c$ from $\xi$ becomes

$$M' = \begin{bmatrix} k_4 k_5 & k_2 k_6 & -k_2 k_4 \\ k_3 k_4 & k_1 k_6 & -k_1 k_6 \\ k_1 k_2 & k_3 k_6 & -k_3 k_6 \end{bmatrix},$$

$$c = \xi \otimes M'.$$

This yields $m^{k_1 k_2 k_3 + k_2 k_4 k_6}$, which is the ciphertext if the constants $k_i$ are chosen to satisfy

$$e = k_1 k_4 k_5 + k_2 k_3 k_6.$$

Such constants, for example, are:

$$k = \begin{bmatrix} 2 & 3641 & 3 & 2^7 & 2^7 & 3 \end{bmatrix}.$$

Using these values, the calculation becomes

$$\xi = \begin{bmatrix} m^{2^3} r_1^{2^{23}} & m^{3^2} r_2^{28} & r_1^{128} r_2^{3} \end{bmatrix}.$$

Since the powers of $r_i$ can be precalculated, only $m^2$ and $m^3$ needs to be calculated in real-time, plus the multiplication of them and their coefficients. This adds up to a total of 4 real-time multiplications to calculate an RSA equivalent signature (using the public exponent $e = 65537$), instead of the 17 multiplications required by fast exponentiation. An optimal $k$ vector can be determined by a finite search; this was performed with the help of [14] and resulted in

$$k_{\text{opt}} = [2 \quad 1283 \quad 3 \quad 4 \quad 13 \quad 17],$$

and the calculation can be written as

$$\xi = \begin{bmatrix} m^{2^3} r_1^{2^{23}} & m^{3^2} r_2^{28} & r_1^{128} r_2^{3} \end{bmatrix}.$$
2.2.2 Decryption

To decrypt the message, first we calculate \( c \) from \( c_i \); in other words, we first reproduce the output of the original RSA encryption. Then the unmodified RSA decryption algorithm can be used to decrypt this new ciphertext. In Section 2.2.1 we have stated that \( c \equiv c \mod M' \) (mod \( n \)). This yields the following equation:

\[ c \equiv c_1^{k_1} c_2^{k_2} (c_3^{-1})^{k_3} (c_4^{-1})^{k_4} \mod n. \]

Here the inverse of \( c_3 \) can be calculated using, for example, the Extended Euclidean Algorithm. Due to the definition of vector \( z \), this becomes:

\[ c \equiv k_1^k k_2^k (m^{k_3} r_1^{k_3}) k_5^k (m^{k_4} r_2^{k_4}) k_6^k - k_2 k_4 \]
\[ = k_1^k k_3 k_4 k_5 k_6 k_7 k_8 k_9 - k_2 k_3 k_4 \]
\[ = m^e \mod n. \]

The last step holds because \( k_i \)'s are chosen in Section 2.2.1 such that \( c = k_1 k_2 k_3 k_4 \). After calculating \( c \), one can simply use the unmodified RSA algorithm, i.e., calculate \( m = c^d \mod n \).

The decryption of this algorithm takes more steps than the original RSA algorithm; therefore, the acceleration is more useful in environments where the encrypting participant has much less computing resources than the decrypting one. Such situation often occurs in bus encoders, sensor networks, and other networks where the participants in a star topology send encrypted messages to the center node. Note that, if hardware accelerators are available for the decryption (but not for the encryption), the calculation of \( c \) can further be accelerated by using three modular exponentiation modules in parallel. Thus, the decryption of a message takes one modular inverse and four modular exponentiations, three of them calculated in parallel.

This algorithm can be used for any \( e \) exponent, provided that one can find a suitable \( k \) vector for it. For efficiency, \( k_1 = 2 \) and \( k_3 = 3 \) can be assumed; thus, only four values remain to be searched. This can be performed by a finite search (4 levels of cycles).

2.3 Security of the System

Attacks of the cryptosystem described in Section 2.2 fall into two categories: either one tries to factor one of the \( c_i \) messages, or one tries to reconstruct \( m \) or \( r_i \) from multiple \( c_i \)'s.

In the first case, one has to solve the factoring problem. If either of the \( r_i \) numbers is small, or has a small factor, this is considered easy. However, many secure random number generators exist that don't yield integers with small factors; and \( r_i \)'s can even be chosen primes. Any algorithm that can factor one \( c_i \) (to primes) without the other two could be used to solve the RSA problem itself: since \( c_i \)'s can be considered as the product of RSA-encrypted messages with a special public exponents\(^1\).

The second case is when an attacker tries to calculate either \( m \) or \( r_i \) from all the \( c_i \)'s. Directly multiplying the powers of \( c_i \)'s, while trying to eliminate \( r_i \)'s will lead to \( m^e \), and not \( m \) due to the conditions on \( k_i \) exponents described in Section 2.2. This is the first step in the decoder, used for extracting the ciphertext from \( c_i \), and does not lead to an attack.

One might try to write \( m^2 \) as the greatest common divisor of \( c_1 \) and \( c_2 \). This is usually performed by running a version of the Euclidean algorithm, which requires an ordered field. Modular fields, however, are not ordered, therefore the algorithm has to branch 3 ways on each comparison. One of these ways is a termination check requiring a non-trivial verification of the result candidate, while the other two are recursions. Such an algorithm is exponential even if the verification is performed in constant time. After finding \( m^2 \), one still has to calculate its modular square root; this is equivalent to breaking the Rabin cryptosystem [10].

A similar method is to try deducing \( r_1 \) from the greatest common divisor of \( c_1 \) and \( c_3 \) (or similarly, \( r_2 \) as \( \gcd(c_1, c_2) \)). If \( \gcd(r_2, m) = 1 \), then

\[ r_2^{\min(k_4, k_5) = \gcd(c_2, c_3)}. \]

Doing the same for \( r_1 \) yields to

\[ r_1^{\min(k_2, k_5) = \gcd(c_1, c_3)}. \]

If \( k_2 \leq k_5 \) and \( k_4 \leq k_6 \), then this reveals both \( m^2 \) and \( m^3 \), and one can calculate \( m \) with an inversion and a multiplication. If either \( k_2 > k_5 \) or \( k_4 > k_6 \), then one acquires only \( r_1^{k_5} \) (or \( r_2^{k_6} \)); thus, an instance of the RSA problem must be solved to get \( r_1 \) (or \( r_2 \)) and thus \( m^2 \) (or \( m^3 \)). Note however, that the greatest common divisor algorithm in a non-Euclidean field, which is required in the first step, has an exponential runtime.

An almost trivial attack is possible if the random numbers are compromised. In this case, one might calculate \( r_1^{-k_5} \) to retrieve \( m^2 \) from \( c_1 \), and \( r_2^{-k_6} \) to retrieve \( m^3 \) from \( c_2 \). Then, it is possible to use the common modulus attack [3, 6]. If the random values are reused for multiple encryptions, and one of the messages is revealed, then it is possible to retrieve \( r_1 \) and \( r_2 \) (and thus all the messages encrypted with the same random values) using a similar method.

The security of the system can be further enhanced if the three parts of the crypto-text are transmitted on distinct channels. If an attacker is only able to intercept messages on up to two channels, then it is theoretically impossible to reconstruct \( c \) and therefore \( m \) cannot be retrieved. This is because either \( r_1 \) or \( r_2 \) will only appear in only one of the messages, so its power can also be considered random. This layout is recommended for sensor networks where non-overlapping directed antennas might be used for the three channels.

\(^1\)For \( c_i \), \( m^2 \) should be factored, which is the discrete square root problem.
3 Conclusions

In this paper we have shown a method to accelerate the real-time behavior of RSA encryption. We achieved a real-time performance of 4 multiplications per signature compared to the 17 for the standard algorithm and default \((e = 65537)\) public key. This is achieved at the cost of precalculations and three messages to be transmitted.

Acknowledgments

The authors wish to thank HSNLAB for the support of the research, and to the anonymous reviewers for valuable comments.

References


Loránd Szőllősi received M.Sc. degree in Technical Informatics in 2005 and Expert-Engineer in Informatics of Banking Business in 2007 at BME. Recently he is working on his Ph.D-thesis about accelerating digital signatures in the Dept. of Telecommunications and Media Informatics (BME), supported by the High Speed Network Laboratory. He gives lectures in the topic of network security.

Tamás Marosits was born in 1972. He received the M.Sc. degree in Electrical Engineering at the BME in 1996. Recently he is working on his Ph.D-thesis (Theme: call acceptance and traffic control algorithms of high speed packet switched info-communication networks) in the Dept. of Telecommunications and Media Informatics (BME). His other research fields are the security and privacy of the info-communication networks and VoIP systems.

Gábor Fehér graduated in 1998 as a Technical Informatics at the Budapest University of Technology and Economics. He received his Ph.D. degree in 2004 in the field of resource control in IP networks. Currently he is an assistant professor of the Dept. of Telecommunications and Media Informatics. He gives lectures in the topics of network and multimedia security. Since 1997 he is a member of the High Speed Network Laboratory of the Department and is involved in several national and international research projects.