cryptography

Article

A Secure Algorithm for Inversion Modulo 2^k

Sadiel de la Fe [1,2,*,†] **and Carles Ferrer** [1]

[1] Department of Microelectronics and Electronic Systems, Universitat Autònoma de Barcelona, 08193 Barcelona, Spain; carles.ferrer@uab.cat

[2] Applus Laboratories, Bellaterra, 08193 Barcelona, Spain

* Correspondence: sadiel.delafe@e-campus.uab.cat; Tel.: +34-935-813-561

† Current address: Applus Laboratories, Carrer de la Font del Carme, Bellaterra, 08193 Barcelona, Spain.

Received: 21 August 2018; Accepted: 12 September 2018; Published: 13 September 2018

Abstract: Modular inversions are widely employed in public key crypto-systems, and it is known that they imply a bottleneck due to the expensive computation. Recently, a new algorithm for inversions modulo p^k was proposed, which may speed up the calculation of a modulus dependent quantity used in the Montgomery multiplication. The original algorithm lacks security countermeasures; thus, a straightforward implementation may expose the input. This is an issue if that input is a secret. In the RSA-CRT signature using Montgomery multiplication, the moduli are secrets (primes p and q). Therefore, the moduli dependent quantities related to p and q must be securely computed. This paper presents a security analysis of the novel method considering that it might be used to compute secrets. We demonstrate that a Side Channel Analysis leads to disclose the data being manipulated. In consequence, a secure variant for inversions modulo 2^k is proposed, through the application of two known countermeasures. In terms of performance, the secure variant is still comparable with the original one.

Keywords: modular inversion; montgomery multiplication; RSA; side channel attack

1. Introduction

Public key cryptographic schemes often require performing modular inversions, which are known to be expensive operations. In RSA, for example, the secret key is obtained through the inversion of the public key. In ECDSA (Elliptic Curve Digital Signature Algorithm), to generate a digital signature, the per-message random secret is inverted after the scalar multiplication. Fermat's, Euler's and Euclidean's methods are the most well-known solutions to compute a multiplicative inverse. Derived from Euclid's method, the Binary Extended Euclidean Algorithm (BEEA) is very efficient as it substitutes multi-precision divisions by right shifts. This is a suitable approach for software and hardware realizations [1]. However, a straightforward BEEA implementation is susceptible to Side Channel Analysis (SCA) [2,3].

Recently, a very efficient algorithm to compute the multiplicative inverse modulo p^k has been introduced by Koç in [4]. The new inversion method has a low computational complexity. This is a clear advantage, since cryptographic implementations usually manipulate large numbers. A special case of the algorithm is the computation modulo 2^k. This is especially useful to compute the required modular inverse in a Montgomery multiplication [5]. The method, however, is clearly not intended to manipulate sensitive data, as it will be analyzed in next sections. Therefore, a straightforward implementation should be avoided, for example, in the Chinese Remainder Theorem variant of RSA (RSA-CRT).

Some other approaches had been previously introduced to compute the inverse modulo 2^k [6,7]. Those algorithms do not seem either intended to manipulate secrets and their performance is lower compared to the new one (see the algorithms comparison given in [4]). Thus, we focus this work in the analysis of the new inversion method, because a secure version of it may be a suitable candidate to be used in low power devices with cryptographic capabilities.

1.1. RSA-CRT with Montgomery Multiplications

RSA is an asymmetric cryptosystem that allows both encryption and signing [8]. The cryptosystem is still a widely used standard, even in financial sector products like smart cards [9]. Despite its age, there are works that recently addressed some security research on RSA implementations [10–12]. RSA signature of message m, by using the private key d can be written as

$$S = m^{d} mod\ N \tag{1}$$

where N is a public modulus compound by the multiplication of two secret primes ($N = p \cdot q$). As N is a large number, the modular exponentiation is a costly operation. RSA-CRT variant is preferred for efficiency reasons. As p and q are both smaller than N, residue-based arithmetic (modulo p and q) allows working with shorter registers, and then, the exponentiation complexity gets reduced by signing the message following

$$S_p = m^{dp} mod\ p \tag{2}$$

$$S_q = m^{dq} mod\ q \tag{3}$$

where d_p and d_q are the residues of the private key modulo p and q, respectively.

In Equations (2) and (3), two partial signatures are obtained. To give a unified result, these values need to be joined. The recombination methods of Gauss and Garner are well known to do that. Garner's recombination (below) is often preferred for being more efficient than Gauss's method.

$$S = S_q + q \cdot (q^{-1} \cdot (S_p - S_q)\ mod\ p) \tag{4}$$

The main advantage of the Montgomery multiplication (see Algorithm 1) is the substitution of divisions by right shifts and modular reductions by truncations. Because of that simplification, this method is commonly used to solve the modular multiplications involved in the exponentiation.

Algorithm 1: MontgomeryMult

```
Input: $a, b, n, r, n'$; such that $n < 2^k$ and $r = 2^k$
Output: $u = a \cdot b \cdot r^{-1} mod\ n$
1. $t = a \cdot b$
2. $m = t \cdot n'\ (mod\ r)$
3. $u = (t + m \cdot n)/r$
4. if $u \geq n$
5.     $u = u - n$
6. return $u$
```

Even when the output is multiplied by r^{-1}, Algorithm 1 works with an n', which is such that $2^k \cdot r^{-1} - n \cdot n' = 1$. Therefore, the calculation of n' can be given by $n' = -n^{-1} mod\ 2^k$. In the case of the standard RSA, the modulus N is public; however, in RSA-CRT the partial exponentiations are calculated using moduli p and q, which are both secrets. The analogue calculations of n' in RSA-CRT would be

$$p' = -p^{-1} mod\ 2^k \tag{5}$$

$$q' = -q^{-1} mod\ 2^k \tag{6}$$

where k is such that $2^{k-1} < p, q < 2^k$. The usage of a non-protected algorithm for the calculation of n' in a standard RSA does not imply any risk. On the contrary, for the RSA-CRT, if p' and q' need to be computed, it should be done more carefully, because the secret primes are directly involved. Actually, if p and q are dynamically masked (p_m and q_m) at each RSA-CRT computation, as it commonly occurs in banking products, the Montgomery constants p'_m and q'_m will be different every time and they should always be computed.

1.2. Our Contributions

In this work, we conduct a security evaluation of the new inversion method proposed by Koç. We demonstrate that the algorithm lacks security countermeasures, and then a straightforward implementation of it may compromise a secret if it is being manipulated. A secure and still efficient variant for the computation of the inverse modulo 2^k is proposed herein. It includes countermeasures that allows handling sensitive data in a safe mode, as needed in the case of RSA-CRT with Montgomery multiplication.

1.3. Paper Organization

This paper is organized as follows: in Section 1 an introduction to the topic is given. Section 2 describes the inversion method under study. In Section 3 a security analysis is conducted for the special case of inversions modulo 2^k, where two vulnerabilities are discussed. In Section 4 the countermeasures to patch the vulnerabilities are described and a SCA-protected variant of the algorithm is presented. Finally, Conclusions and References are listed.

2. On a New Algorithm for Inversion Modulo p^k

The algorithm introduced in [4] focuses on the need for the public key cryptographic schemes to perform modular inverse operations. The new method to compute $x = a^{-1} mod\ p^k$ seems to be quite efficient and it works for any p and any k. The assumptions to perform the computation are:

- p is a prime
- k is a positive integer
- $gcd(a, p) = 1 \quad (1 < a < p^k)$

p is usually a small number (commonly 2 or 3), thus the computation at step 1 is expected to be easily performed. In fact, for the case of $p = 2$, the computation of c is trivial. A better comprehension of Algorithm 2 and its demonstrations, can be obtained from the work in [4].

Algorithm 2: Modular Inverse [mod p^k]

```
Input: $a$, $p$ and $k$; such that $gcd(a, p) = 1$ and $a < p^k$
Output: $x = a^{-1} mod\ p^k$
  1. $c = a^{-1} mod\ p$
  2. $b_0 = 1$
  3. for $i = 0$ to $k - 1$
  4.     $X_i = c \cdot b_i\ mod\ p$
  5.     $b_{i+1} = (b_i - a \cdot X_i)/p$
  6. return $x = (X_{k-1} \ldots\ldots X_1 X_0)_p$
```

Special case p = 2

In the previous section, we discussed the Montgomery constant computation, and it was highlighted the need for the modular inverse in that process. As from Equations (5) and (6) for the RSA-CRT cryptosystem, the said inverse is computed modulo 2^k which is a particular case of modulo p^k where $p = 2$.

Algorithm 2 can be reduced if $p = 2$ to compute the inverse modulo 2^k. In this case, the computation of $x = a^{-1} mod\ 2^k$ requires that $gcd(a, 2^k) = 1$, thus a must be an odd number and then $c = 1$.

From Algorithm 3 one appreciates that the operation at step 3 is trivial, as it only requires checking the LSB of b_i. On the other hand, the returned value in x is binary. The simplified algorithm for $p = 2$ follows

Algorithm 3: Modular Inverse [mod 2^k]

```
Input: $a$ and $2^k$; such that $a < 2^k$ and $a$ is odd
Output: $x = a^{-1} mod\ 2^k$
  1.  $b_0 = 1$
  2.  for $i = 0$ to $k - 1$
  3.      $X_i = b_i\ mod\ 2$
  4.      $b_{i+1} = (b_i - a \cdot X_i)/2$
  5.  return $x = (X_{k-1} \ldots\ldots X_1 X_0)_2$
```

3. Security Analysis for $p = 2$

Side Channel Analysis techniques have been introduced in the late 1990s by Kocher [13]. An effective attack leads to disclose sensitive data being manipulated by a cryptographic device, through the leakage associated with its power consumption. One of such techniques is the Simple Power Analysis (SPA). Through an SPA, one can observe the sequence of operations, or one can even distinguish an operation from another one by the differences in their power consumption patterns. The observation of a power consumption trace may also give a clue of the operation latency, which is closely related to the bit length of the operands. This applies mainly for sequential operations. In the following subsections we describe two vulnerabilities found in the inversion algorithm under analysis, that impede a safely manipulation of secrets.

3.1. Asymmetric Iterations

It is well known that the Square-and-Multiply method to compute $y = g^k$ allows to recover k through an SPA, due to a difference in the operations performed whether $k = 0$ or $k = 1$. The Montgomery ladder exponentiation solves that issue by performing the same operations at every iteration of the algorithm [14].

A similar issue has been detected in the inversion method under analysis in this work. It allows a straightforward SPA, which leads to an easy recovery of the operation result, and in consequence, the input data is disclosed. As from the previous section, the modular inverse of the input a, obtained through Algorithm 3 is formed by $x = (X_{k-1} ... X_1 X_0)_2$; where $X_i \in [0, 1]$. Furthermore, the intermediate result $b_i - a \cdot X_i$ at step 4 is always divisible by 2. At step 4, besides the multiplication $a \cdot X_i$, two other operations can be distinguished: a subtraction and a division by 2. The division can be performed as a right shift because the result of the subtraction is always divisible by 2.

Regarding the subtraction, this may or may not be computed. One can see that if $X_i = 0$, then $a \cdot X_i = 0$, and then the subtraction $b_i - a \cdot X_i$ becomes $b_i - 0$. In a straightforward implementation of the original algorithm, the developer may choose to obviate the subtraction if $X_i = 0$. We recall that the original work does not refer to any SCA protection to keep the input data safe, thus we believe the author did not consider a scenario with a secret input. If the subtraction is not performed, a significant difference in the execution flow exists depending on the X_i value. In summary

$$X_i = 0 \;\rightarrow\; b_{i+1} = (b_i - a \cdot X_i)/2 = b_i/2$$

$$X_i = 1 \;\rightarrow\; b_{i+1} = (b_i - a \cdot X_i)/2 = (b_i - a)/2$$

Such a data-dependant characteristic could be distinguished in a power consumption trace of the algorithm execution. It then leads to a straightforward SPA where the modular inverse of the secret could be directly recovered. Once the modular inverse is recovered, it is then trivial to obtain the input by computing $a = x^{-1} mod\ 2^k$. If a was a secret, as it is the case in the Montgomery constants computation for RSA-CRT, this would imply a critical security issue.

However, the developer may choose a more regular implementation by always computing the subtraction. In this case, there are two possibilities: if $X_i = 0$, the subtraction $b_i - 0$ is computed, while if $X_i = 1$, it will be computed $b_i - a$. The operands b_i and a are large integers since the second iteration, in the context of the Montgomery constants computation for RSA-CRT. It makes the subtraction $b_i - a$ highly susceptible of having lots of carry bits propagation. This effect has a negative impact on the latency of the addition/subtraction. While in $b_i - 0$ the carry propagation is null, in $b_i - a$ the carry propagation varies making that operation longer in time. This should be enough to apply a Timing Attack to distinguish one operation from the other, which directly leads to infer the values of the related X_i.

3.2. Operations Latency

The latency of the arithmetic operations is closely related to the data length of the operands, especially in software implementations. In RSA, for example, the exponentiation latency is expected to be proportional to the key length. In the case of additions/subtractions, they both commonly require to manage a carry bit that can be generated at each bit-bit operation. Therefore, the carry chain is as long as the operands, and it determines the whole operation latency.

Let us say, for example, that the evenness of an operand determines the next operation where it will be involved, and the said operation impacts on the operand's bit length. If that quantity is further added or subtracted from a constant value and this sequence is performed in a loop, the addition/subtraction latency might experiment variations at each iteration, as a consequence of the carry chain modification. If an adversary is able to identify the additions/subtractions through an SPA and measure those variations, then the operand's evenness (its Least Significant Bit—LSB—) might be traced back.

From Algorithm 3, one can see that the subtraction at the step 4 depends on b_i and a. The value of a is invariant throughout the whole operation, while b_i does varies. In fact, the value of b_i is strongly dependent on X_i. If $X_i = 0$, then b_{i+1}, computed at iteration i, yields $b_i/2$. For consecutive $X_i = 0$, the respective b_{i+1} are always smaller by a factor of 2. On the other hand, considering $a < 0$ (as required in the Montgomery constant computation), it can be demonstrated that b_{i+1} tends to a for consecutive $X_i = 1$.

Let's have $X_i = X_{i+1} = 1$. The correspondent calculations of b_i and b_{i+1} follow

$$b_i = b_{i-1}/2 + a/2 \tag{7}$$

$$b_{i+1} = b_{i-1}/4 + 3a/4 \tag{8}$$

According to the right side of Equation (8) and comparing it with the right side of Equation (7), the second fraction (which depends on a) in (8) is greater and it approaches more to a. Meanwhile, the first fraction is halved and tends to zero. Thus, it makes b_{i+1} closer to a rather than to b_i. Something similar occurs when $X_i = 0$ and $X_{i+1} = 1$.

In summary, we might then expect to observe in a power trace, a continuous decreasing latency in the addition/subtraction for consecutive iterations where $X_i = 0$; while the latency would tend to increase for continuous $X_i = 1$ or even for transitions from $X_i = 0$ to $X_{i+1} = 1$.

The differences in the execution flow for the cases $X_i = 0$ and $X_i = 1$, presented in the previous section, are enough to perform an SPA on Algorithm 2. Thus, a timing analysis for this purpose is not necessary; however, in order to design a countermeasure to overcome such data-dependent vulnerability, the issue on the operations timing has to be taken into account.

4. A Secure Method for Inversion Modulo 2^k

Considering the issues found in the previous section, a secure variant of Algorithm 3 is proposed herein (See Algorithm 4).

Algorithm 4: Secure Modular Inverse [mod 2^k]

```
Input: $a$ and $2^k$; such that $a < 2^k$ and $a$ is odd
Output: $x = a^{-1} mod\ 2^k$
 1.  $b_0 = 1 + 2^k$
 2.  for $i = 0$ to $k - 1$
 3.      $b_i = b_i + 2^k$
 4.      $X_i = b_i\ mod\ 2$
 5.      if $X_i = 0$
 6.          $b_{i+1} = b_i/2$
 7.          $b_f = (b_i - a)/2$
 8.      else [if $X_i = 1$]
 9.          $b_f = b_i/2$
10.          $b_{i+1} = (b_i - a)/2$
11.  return $x = (X_{k-1} \ldots\ldots X_1 X_0)_2$
```

Our new variant resists both SPA and Timing Attacks with a minimum overhead; therefore, it can be used to obtain the multiplicative inverse of odd secrets modulo 2^k. Moreover, the low complexity of the countermeasures applied makes this algorithm suitable to be implemented in low power devices.

In our algorithm, two conditional branches have been included for the cases of $X_i = 0$ and $X_i = 1$ respectively. They both compute the same operations, in the same order, with the current values of b_i. Dummy operations have been introduced in both branches to balance them. The variable b_f holds the useless result derived from the dummy operation. This simple countermeasure follows the same strategy as in the Montgomery ladder and impedes the recognition of X_i through an SPA.

Moreover, the subtraction $b_i - a$ is executed for all X_i, and even in the fake case, the operation manipulates the actual value of b_i. Following the analysis in Section 3.2, the subtraction latency depends on b_i, which varies in every iteration depending on X_i. Thus, it can be said that, for every iteration, the latency of $b_i - a$ depends on X_i. Therefore, a constant bit length of b_i (that implies a constant carry chain length in $b_i - a$) is a must so that the subtraction has the same latency throughout the whole execution.

Previously, it was seen that b_{i+1} decreases if $X_i = 0$, and that b_{i+1} tends to a for continuous $X_i = 1$. As $a < 2^k$, it has at most k bits and so has b_i. When $a < 0$ (as in the Montgomery constant computation), the operation $b_i - a$ becomes an addition. In such cases b_{i+1} might have at most $k + 1$ bits, considering the carry.

To get X_i at step 4, the evenness of b_i is evaluated. If an even value is added to b_i, it will not affect b_i's evenness. On the other hand, if a number $v = 2^{k+1}$ is added to b_i, it will not affect the first $k + 1$ bits of the operation result at steps 7 and 10.

Following this reasoning, X_i at step 4 could be evaluated with $b_i + 2^{k+1}$ and then $b_{i+1} = (b_i + 2^{k+1} - a)/2$. Please note that, as the algorithm evaluates only k bits of the resultant b_i, the Most Significant Bit (MSB), at position $k + 1$, will have no effect on the result. Table 1 gives a detailed view of this.

In the algorithm, the loop first gets a b_i of $k + 2$ bits, where its MSB is '1'. After the subtractions at steps 7 and 10, a division by 2 is carried out. It makes the MSB to shift one bit right to occupy the 2^k position. This is compensated in the next iteration, at the step 3, by adding 2^k so that b_i has $k + 2$ bits again; it is, with the MSB in the 2^{k+1} position.

Table 1. Addition $a + b_i$ not affected by the summand 2^{k+1}.

	2^{k+1}	2^k	2^{k-1}	...	2^1	2^0
a	-	-	1	...	x	x
b_i	-	-	1	...	x	x
$a + b_i$	-	1	x	...	x	x
v	1	0	0	...	0	0
$a + b_i + v$	1	1	x	...	x	x

The starting point was that b_i different bit lengths may cause a latency variation in the subtractions, and it may lead to infer the previous value of X_i. By adding 2^{k+1} to b_i, it guarantees that subtractions at steps 7 and 10 are always performed with operands of constant bit length. At step 3, b_i will always have $k + 1$ bits, so the addition $b_i = b_i + 2^k$ will always be performed with constant bit length operands too. Furthermore, the right shifts at steps 6 and 9 will be carried out with b_i having fixed $k + 2$ bits length.

Secure Variant Overhead

The proposed variant herein implies no significant overhead in comparison with the original algorithm. The count of operations of the original inversion method yields one addition. This is true if we dismiss the b_i evenness check and the division by 2. Our algorithm does not add any further addition. In fact, the operation at step 3 may be coded in a few ways, but in any case, it is only needed to manipulate the most significant byte of b_i to set the bit in the 2^{k+1} position (see Table 1).

5. Conclusions

The analysis performed on the original inversion algorithm for modulo 2^k led us to establish a direct relationship between the output data bits and the execution flow. In consequence, we demonstrated that the modular inverse of a secret could be revealed by an SPA conducted on a single power consumption trace.

A timing analysis was also carried out, specifically on the subtraction operation. It was concluded that the bit length of b_i may affect the latency of subtractions, and because the bit length is related to the factor X_i, a direct relationship could also be established between the sensitive output data and the latency of the subtractions.

Having this into account, a secure variant of the original inversion algorithm was proposed. By solving the security issues described, the protected algorithm allows to manipulate secret values, as it is the case in the Montgomery constants computation for RSA-CRT. The secure method is resistant to SPA and Timing attacks. Furthermore, the overhead of the applied countermeasures implies no significant lack of performance respect to the original algorithm.

Author Contributions: Conceptualization, Formal Analysis and Writing: S.d.l.F. Supervision: C.F.

Funding: This research received no external funding.

Acknowledgments: The authors wish to thank Alejandro Cabrera Aldaya for his helpful comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Stein, J. Computational problems associated with Racah algebra. *J. Comput. Phys.* **1967**, *1*, 397–405. [CrossRef]
2. Acıiçmez O.; Koç, Ç.K.; Seifert, J.-P. On the power of simple branch prediction analysis. In Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security, ASIACCS'07, Singapore, 22–27 March 2007; ACM: New York, NY, USA, 2007; pp. 312–320.
3. Cabrera Aldaya, A.; Cabrera Sarmiento, A.J.; Sánchez-Solano, S. SPA vulnerabilities of the binary extended Euclidean algorithm. *J. Cryptogr. Eng.* **2017**, *7*, 273–285. [CrossRef]

4. Koç, Ç.K. A New Algorithm for Inversion Mod p^k. IACR Cryptology ePrint Archive. 2017. Available online: https://eprint.iacr.org/2017/411.pdf (accessed on 12 September 2018).
5. Montgomery, P.L. Modular multiplication without trial division. *Math. Comput.* **1985**, *44*, 519–521. [CrossRef]
6. Dussé, S.R.; Kaliski, B.S., Jr. A cryptographic library for the Motorola DSP56000. In *Advances in Cryptology—EUROCRYPT 90*; Damgård, I.B., Ed.; Springer: Berlin/Heidelberg, Germany, 1990; pp. 230–244.
7. Arazi, O.; Qi, H. On calculating multiplicative inverses modulo 2^m. *IEEE Trans. Comput.* **2008**, *57*, 1435–1438. [CrossRef]
8. Rivest, R.L.; Shamir, A.; Adleman, L.M. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
9. EMVCO. Integrated Circuit Card Specifications for Payment Systems (Book 2—Security and Key Management). 2011. Available online: https://www.emvco.com (accessed on 12 September 2018).
10. Witteman, M. A DPA Attack on RSA in CRT Mode. Riscure Technical Report. 2012. Available online: http://www.riscure.com (accessed on 12 September 2018).
11. Fouque, P.A.; Guillermin, N.; Leresteux, D.; Tibouchi, M.; Zapalowicz, J.C. Attacking RSA-CRT signatures with faults on Montgomery multiplication. *J. Cryptogr. Eng.* **2013**, *3*, 59–72. [CrossRef]
12. Kiss, A.; Kramer, J.; Rauzy, P.; Seifert, J.P. Algorithmic countermeasures against fault attacks and power analysis for RSA-CRT. In Proceedings of the International Workshop on Constructive Side-Channel Analysis and Secure Design, Graz, Austria, 14–15 April 2016; pp. 111–129.
13. Kocher, P.; Jaffe, J.; Jun, B. Differential power analysis. In *Advances in Cryptology (CRYPTO'99)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1666, pp. 388–397.
14. Montgomery, P.L. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comput.* **1987**, *48*, 243–264. [CrossRef]