Improved Shamir's CRT-RSA Algorithm: Revisit with the Modulus Chaining Method

Seungkwang Lee, Dooho Choi, and Yongje Choi

RSA signature algorithms using the Chinese remainder theorem (CRT-RSA) are approximately four-times faster than straightforward implementations of an RSA cryptosystem. However, the CRT-RSA is known to be vulnerable to fault attacks; even one execution of the algorithm is sufficient to reveal the secret keys. Over the past few years, several countermeasures against CRT-RSA fault attacks have tended to involve additional exponentiations or inversions, and in most cases, they are also vulnerable to new variants of fault attacks. In this paper, we review how Shamir's countermeasure can be broken by fault attacks and improve the countermeasure to prevent future fault attacks, with the added benefit of low additional costs. In our experiment, we use the side-channel analysis resistance framework system, a fault injection testing and verification system, which enables us to inject a fault into the right position, even to within 1 μs. We also explain how to find the exact timing of the target operation using an Atmega128 software board.

Keywords: CRT-RSA, fault attack, countermeasure.

I. Introduction

The RSA algorithm has been used as an important cryptographic tool for authentication, signature generation, and verification. This is based on the presumed difficulty of factoring large integers—that is, the factoring problem. Despite the reliability of RSA, however, improvements in its performance remain a challenging research topic owing to its large size modulus. One solution is to adopt the Chinese remainder theorem (CRT) to RSA, which provides approximately a four-fold increase in performance. Additional savings in memory space is also possible owing to the reduced size of the modulus. These are the main reasons why CRT has been used with RSA (CRT-RSA) in smart cards with limited resources. However, injecting a fault (or faults), such as a clock glitch, power glitch, or light, during the computation of CRT-RSA and observing its faulty output gives the attacker information about secret keys [1]; such attacks are known as fault attacks. In contrast to power analysis attacks or brute force attacks against cryptographic algorithms, fault attacks require only one (or a few) fault injections and post-processing algorithms to find the secret keys. Among various fault attacks, some are practical, while others are only theoretically possible. Researchers have also studied countermeasures against fault attacks. As expected, the simplest way is to re-compute the signature and compare the two outputs, but this is not an efficient method. A challenging research topic is how to provide a secure and computationally efficient CRT-RSA algorithm. However, practical and secure solutions are hard to find; most of the countermeasures involve additional exponentiations or inversions [2]–[7], and the majority of those are themselves also susceptible to fault attacks [4], [8]–[11].

We revisit Shamir's countermeasure and show what types of

Manuscript received Apr. 23, 2013; revised Nov. 11, 2013; accepted Nov. 27, 2013.

This work was supported by the K-SCARF project, the ICT R&D program of ETRI (Research on Key Leakage Analysis and Response Technologies).

Seungkwang Lee (phone: +82 42 860 1437, skwnag@etri.re.kr), Dooho Choi (corresponding author, dhchoi@etri.re.kr), and Yongje Choi (choiyj@etri.re.kr) are with the SW · Content Research Laboratory, ETRI, Daejeon, Rep. of Korea.

fault attacks can be applied to this algorithm in practice. We then propose an improved countermeasure against fault attacks without involving additional heavyweight operations such as exponentiations or inversions. The remainder of this paper is organized as follows: we first introduce some basic concepts, including Shamir's countermeasure, and then look at fault attacks that can be practically applied to Shamir's countermeasure in section II. We next propose a new CRT-RSA algorithm in section III, and then we demonstrate its security against fault attacks in section IV. We describe the experiments used to verify the security of the proposed countermeasure and show the experimental results in section V. Finally, we conclude the paper in section VI.

II. Preliminaries

In this section, we introduce some basic concepts and notations that will be used throughout the paper and briefly explain the RSA algorithm. Also included in the explanation is the CRT, which is used to speed up RSA.

1. The RSA Algorithm

RSA, which is named after its inventors, Rivest, Shamir and Adleman, is the first public-key cryptosystem known to be suitable for both digital signatures and encryption. The details of the parameters, including the public and private key pairs, are as follows:

- Generating two large primes, p and q, in equal size so that their product $N = p \cdot q$ has the bit length required by the cryptosystem.
- Compute $N = p \cdot q$ and $\varphi(p \cdot q)$, where φ denotes Euler's totient function.
- Choose a number e, $1 < e < \varphi(p \cdot q)$, such that gcd $(e, \varphi(p \cdot q)) = 1$.
- Find another number d, $1 < d < \varphi(p \cdot q)$, such that $(e \cdot d) - 1$ is divisible by $\varphi(p \cdot q)$.

The two numbers e and d, are called public and private exponents, respectively, and a public key is a pair of (N, e); a private key is (N, d). The factors p and q, may be destroyed or kept secret. It is currently intractable to find the private key d from the public key (N, e). If it is possible to factorize N into p and q, one can obtain the private key d. For this reason, the security of the RSA algorithm is based on the difficulty of factoring large numbers. Algorithm 1 shows the process used for making digital signatures using RSA. Exponentiations and modulo computations are the main operations used in the algorithm and are sometimes too heavyweight to be used in resource-limited devices.

Algorithm 1: RSA digital signing
Input: message msg, private key (N, d).
Output: signature $S = m^d \bmod N$.
1 **begin**
2 Create a message digest to be sent.
3 Represent the digest as an integer m, $0 < m < N-1$.
4 $S \leftarrow m^d \bmod N$.
5 Return S.
6 **end**

2. CRT-RSA

For better computation speed and efficiency, RSA can use the CRT. CRT-RSA provides approximately a four-fold faster computation than directly computing $S = m^d \bmod (p \cdot q)$. The additional benefit is that the two modular exponentiations in CRT-RSA use smaller exponents and smaller moduli, thereby reducing resource consumption.

The following explains the basic concept of CRT. Suppose $n_1, n_2, \ldots, n_k$ are positive integers that are pairwise coprime. Then, for any given set of integers $a_1, a_2, \ldots, a_k$, there exists an integer x satisfying the following simultaneous congruences:

$$\begin{aligned} x &\equiv a_1 \pmod{n_1}, \\ x &\equiv a_2 \pmod{n_2}, \\ &\vdots \\ x &\equiv a_k \pmod{n_k}. \end{aligned}$$

In addition, all solutions x are congruent modulo to the product $N = n_1 \times n_2 \times \cdots \times n_k$. Thus, $x \equiv y \pmod{n_i}$, for $1 < i < k$, if and only if, $x \equiv y \pmod{N}$.

This theorem provides a way to improve the performance of RSA. Unlike RSA, CRT-RSA uses p and q, and three additional secrets d_p, d_q, and i_q—where d_p and d_q are known as CRT exponents and i_q as a CRT coefficient. These values are obtained by computing the following:

$$\begin{aligned} d_p &= d \bmod (p-1), \\ d_q &= d \bmod (q-1), \\ i_q &\Rightarrow q \cdot i_q \bmod p = 1. \end{aligned}$$

Given the quintuple (p, q, d_p, d_q, i_q), CRT-RSA can be represented as shown in Algorithm 2.

Algorithm 2: CRT-RSA
Input: message digest m, private key p, q, d_p, d_q, i_q.
Output: signature S.
1 **begin**
2 $S_p \leftarrow m^{dp} \bmod p$.
3 $S_q \leftarrow m^{dq} \bmod q$.
4 Combine two S_p and S_q using Garner's recombination algorithm as follows:
$S = (((S_p - S_q) \bmod p) \cdot i_q \bmod p) \cdot q + S_q$.
5 Return S.
6 **end**

We note that Gauss' recombination, described below, consumes more memory space than Garner's recombination

$$S = (S_p \cdot q \cdot (i_q \bmod p) + S_q \cdot p \cdot (i_p \bmod q)) \bmod N.$$

Owing to the reduced size of the modulus, CRT-RSA provides two noticeable enhancements: a faster computation speed, and less required memory space. As a cryptographic algorithm, however, CRT-RSA has a critical drawback as it is known to be susceptible to a simple fault attack called a Bellcore attack, which reveals the secret prime factors by inserting a single fault. Simply speaking, fault attacks refer to malicious behaviors used to change the normal executions of a chip by inducing an exploitable fault. In doing so, an attacker can then obtain useful information that will assist them in the revealing of secrets from such hardware devices. A Bellcore attack [12], the most well-known fault attack on CRT-RSA, enables an attacker to reveal the secret prime factors by inducing a single fault on a chip. Suppose that by some event, a fault occurs only during the computation of S_p. We let $\hat{S}_p$ and $\hat{S}$ denote a faulty S_p and a faulty signature, respectively. We then know the following:

$$S \not\equiv \hat{S} \bmod p, \text{ but } S \equiv \hat{S} \bmod q,$$

which gives us

$$\gcd((S - \hat{S}) \bmod N, N) \Longleftrightarrow \gcd((m - \hat{S}^e) \bmod N, N) = q.$$

Thus, the attacker can easily factorize N. There are a variety of countermeasures for protecting against a Bellcore attack. In the next section, we introduce Shamir's countermeasure [13] and several fault attacks that can be practically applied to this countermeasure.

3. Shamir's Countermeasure and Fault Attacks

To protect against the fault attack described in the previous section, Shamir proposed a CRT-RSA algorithm, shown in Algorithm 3. Unlike recently published CRT-RSA countermeasures, this countermeasure requires d as an input. Thus, this can be a burden for the overall computation.

Algorithm 3: Shamir's countermeasure
Input: message digest m, private key p, q, d, i_q.
Output: signature $S = m^d \bmod N$.

```
1  begin
2      Generate a random prime r.
3      S_pr ← m^(d mod φ(p·r)) mod p·r.
4      S_qr ← m^(d mod φ(q·r)) mod q·r.
5      if S_pr ≢ S_qr mod r then
6          Return error.
7      end
8      S_p ← S_pr mod p and S_q ← S_qr mod q.
9      Recombine S_p and S_q as explained previously.
10     Return S.
11 end
```

Since p, q, and r are prime numbers, we know that

$$S_{pr} \equiv S_{qr} \bmod r,$$

provided that faults are never injected. As it is, noting $\hat{S}_{pr}$ as the faulty value of S_{pr} and $|r|$ as the bit size of r, we have the following:

$$\Pr[\hat{S}_{pr} \equiv S_{qr} \bmod r] \approx 2^{-(|r|-1)} \ln 2.$$

Based on this probability, if any fault is injected during the computation of S_{pr} or S_{qr}, an error message must be returned with a probability of about $1 - (2^{-(|r|-1)} \ln 2)$.

However, this countermeasure has several drawbacks. First, a fault induced while p (or q) is accessed to compute $p \cdot r$ (or $q \cdot r$, respectively) is not detected. This is based on the assumption that p is likely to be reloaded when needed owing to the limited amount of registers in smart cards. With a faulty p (or a faulty q), a Bellcore attack reveals the secrets. Second, the computations below are not protected

$$S_p \leftarrow S_{pr} \bmod p \text{ and } S_q \leftarrow S_{qr} \bmod q.$$

This reveals the same security hole of the straightforward CRT-RSA on a Bellcore attack. Similarly, the recombination of S_p and S_q is never protected; and as a result, it is possible to inject a fault to i_q without being noticed. Let $\hat{i}_q$ denote a faulty value of i_q, we then have the following:

$$\hat{S} = (((S_p - S_q) \bmod p) \cdot \hat{i}_q \bmod p) \cdot q + S_q$$

and

$$S \not\equiv \hat{S} \bmod p, \text{ but } S \equiv \hat{S} \bmod q.$$

This enables an attacker to perform a Bellcore attack.

We do not take into account an attack on the security comparison. For these kinds of attacks, the attacker has to disturb two precise parts of the computation to bypass the checking procedure: a) a temporary value or an operation to be disturbed and b) the coherence test to bypass. The latter will be possible if the attacker modifies the zero flag of the status register so as to bypass the security comparison. However, it is strongly assumed that sensitive registers are unprotected by redundancy mechanisms such as hardwired checksums or error-correcting codes. In addition, we set aside zero-value attacks in which the attacker is supposed to set one of the target buffers to zero during the execution of the exponentiation. To the best of our knowledge, it is not possible in practice to set a large buffer to zero [14].

Thus far, several variants of Shamir's countermeasure have been proposed. In most cases, they tend to involve additional exponentiations or inversions; thus, resulting in performance

degradation. Furthermore, most of them are still susceptible to fault attacks.

III. New CRT-RSA Algorithm Based on Modulus Chaining for Protecting against Fault Attacks

In this section, we propose a new CRT-RSA algorithm withstandable to fault attacks. Our algorithm, based on Shamir's countermeasure, performs additional security-purpose operations; thus, making up for the vulnerabilities of Shamir's countermeasure. From the practical standpoint of smart cards, it is a noticeable aspect that our algorithm does not need additional exponentiations or inversions from Shamir's algorithm, which results in acceptable performance and resource requirement. We exclusively focus on how to offer a fault-infective CRT-RSA [15], thereby preventing a Bellcore attack. To that end, an error induced by a fault attack must lead to a fault-infective CRT computation on both S_p and S_q, or on the overall computation of S. Consequently, any kind of fault is expected to result in

$$S \not\equiv \hat{S} \bmod p \text{ and } S \not\equiv \hat{S} \bmod q.$$

In particular, we aim to use a special purpose fault-infective computation on a secret modulus such that a fault induced while a secret prime is being loaded spreads throughout the other secret primes. For this purpose, we constructed a modulus chaining where all of the secret primes are tangled up together. This modulus chaining consists of two main concepts: PUSH and POP. In Algorithm 4, PUSH is a procedure adding input values to the target variable sum, where the addition is performed by XORing. In Algorithm 5, POP, on the other hand, outputs the value of a specific variable as follows. Given $sum = \alpha \oplus \beta \oplus \gamma$, for example, [POP α] performs the following:

1) Loads β and γ from memory.
2) Computes $T_{sum} = \beta \oplus \gamma$.
3) Computes $value_\alpha = sum \oplus T_{sum}$.
4) Updates sum using T_{sum} and $value_\alpha$, that is,
   $sum = T_{sum} \oplus value_\alpha$.
5) Returns $value_\alpha$.

What is important here is that [POP α] does not load α from memory but instead loads β and γ, and then it computes $value_\alpha$ from sum. In addition, [POP α] updates $sum = value_\alpha \oplus \beta \oplus \gamma$. These contribute to the following facts. First, if any fault is induced while β or γ is accessed, or while T_{sum} is computed, it results in a faulty $\hat{T}_{sum}$ and consequently produces faulty $\hat{sum}$ and faulty $\hat{value}_\alpha$. Consequently, subsequent POPs performed on this faulty $\hat{sum}$ result in faulty outputs. Moreover, we verify the integrity of sum in the later part of the algorithm so that the faulty $\hat{sum}$ returns an error code. Second, injecting faults while $value_\alpha$ or sum is computed has the same result as the previous case. This has an influence on not only $value_\alpha$ but also on the subsequent outputs of POPs. Owing to these properties, the attacker is unable to succeed in a Bellcore attack if faults are injected while PUSHs or POPs are executed. The details of the proposed algorithm are represented in Algorithm 6.

Algorithm 4: PUSH $x, y, \cdots$.
Input: variable $x, y, \cdots$.
1 **begin**
2 $sum \leftarrow x \oplus y \oplus \cdots$.
3 $W \leftarrow W \cup \{x, y, \cdots\}$.
4 **end**

Algorithm 5: POP x
Input: variable x,
Output: the value of x.
1 **begin**
2 $T \leftarrow W - \{x\}$.
3 $T_{sum} \leftarrow \bigoplus_{t \in T} t$.
4 $value_x \leftarrow sum \oplus T_{sum}$.
5 $sum \leftarrow T_{sum} \oplus value_x$.
6 return $value_x$.
7 **end**

Algorithm 6: Proposed algorithm
Input: message digest m, private key p, q, d, i_q.
Output: signature $S = m^d \bmod N$.
1 **begin**
2 Generate a random prime r.
3 [PUSH p, q, and r].
4 $d_{pr} \leftarrow d \bmod \varphi([\text{POP } p] \cdot [\text{POP } r])$.
5 $d_{qr} \leftarrow d \bmod \varphi([\text{POP } q] \cdot [\text{POP } r])$.
6 $\overline{p} \leftarrow ([\text{POP } p] \cdot [\text{POP } r])$.
7 $\overline{q} \leftarrow ([\text{POP } q] \cdot [\text{POP } r])$.
8 $S_{pr} \leftarrow m^{d_{pr}} \bmod \overline{p}$.
9 $S_{qr} \leftarrow m^{d_{qr}} \bmod \overline{q}$.
10 **if** $S_{pr} \not\equiv S_{qr} \bmod [\text{POP } r]$ **then**
11 Return error.
12 **end**
13 $S_p \leftarrow S_{pr} \bmod [\text{POP } p]$.
14 $S_q \leftarrow S_{qr} \bmod [\text{POP } q]$.
15 $S \leftarrow (((S_p \leftarrow S_q) \bmod p) \cdot i_q \bmod p) \cdot q + S_q$.
16 **if** $(S \not\equiv S_{pr} \bmod [\text{POP } p]$ **or**
17 $(S \not\equiv S_{qr} \bmod [\text{POP } q])$ **then**
18 Return error.
19 **end**
20 $check \leftarrow sum \bigoplus_{w \in W} w$.
21 **if** $check \neq 0$ **then**
22 Return error.
23 **end**
24 Return S.
25 **end**

Table 1. Comparison of previous countermeasures and our own.

	Time complexity [16]
Aummuler el al. [4]	$4(k+l)^3+2k^2+4kl$
Boscher el al. [17]	$4k^3+11k^2$
Ciet and Joye [18]	$(4k+1)(k+l)^2+4l^3+k^2+l^2+3kl+$inversion
Giraud [14]	$4k(k+l)^2+5k^2+2kl$
Our CRT-RSA	$2(k+l)^3+2k^2+2kl$

On the assumption that each prime used in the algorithm is reloaded every time—owing to the limited number of registers in smart cards—we perform PUSH p, q, and r in the beginning and obtain the value of each prime through POP instead of accessing the primes directly. As previously pointed out, Shamir's countermeasure does not protect computations from obtaining S_p, S_q, or the CRT recombination. For this reason, we check the congruence relations on S, S_{pr}, and p as well as on S, S_{qr} and q after finishing the CRT recombination. The value of *check* must be confirmed as an integrity check to make sure the POPs have been performed without disturbance by faults. Based on Shamir's countermeasure, we place the following additional operations in the algorithm.

- Two congruence modulo operations.
- One simple comparison between *check* and 0.
- Modulus chaining: 1 PUSH and 13 POPs.

It is worth noting that PUSH consists of two XORs, POP, and four XORs. Hence, PUSH/POP impose 54 XORs in total. It is also important to note that no inversions or exponentiations are additionally used from Shamir's countermeasure. Table 1 compares the computational cost of previous countermeasures and our own countermeasure for CRT-RSA, where k is the bit length of secret primes, and l is the bit length of a random prime. Our CRT-RSA algorithm requires two exponentiations of a $(k + l)$-bit modulus with a $(k + l)$-bit exponent, one CRT recombination with two multiplications of two k-bit numbers and two multiplications of k-bit and l-bit numbers. Therefore, the final time complexity of CRT-RSA with our countermeasure is $2(k + l)^3 + 2k^2 + 2kl$, excluding the cost for extra operations (that is, addition and logical operations). We note that our CRT-RSA has almost the same level of complexity with Shamir's countermeasure; the additional operations stated earlier do not impose a noticeable increase in the computational cost.

IV. Security Analysis

We now analyze the security of the proposed algorithm against the fault attacks described in section II. Our analysis does not include all kinds of fault attacks like permanent faults, where some parameters may be permanently corrupted or damaged by some serious environmental factors. The first type of fault is induced while a prime is accessed. As shown in Algorithm 4, there are three primes in the algorithm: p, q, and r. They are loaded at PUSH or POP, except for the CRT recombination. If a fault is injected while a specific prime, say p for example, is accessed, then the following analysis shows the reason why the algorithm is resistant to a fault attack: except for the recombination, p is loaded at PUSH (p, q, r), POP q, and POP r. Let $\hat{p}$ denote a faulty p. If a fault is injected during the execution of PUSH, we then have $\hat{sum} = \hat{p} \oplus q \oplus r$. The faulty $\hat{sum}$ has dependency upon subsequent POPs as follows:

$$\text{POP } p \Rightarrow \hat{sum} \oplus q \oplus r:$$
$$\Rightarrow \hat{p} \oplus q \oplus r \oplus q \oplus r = \hat{p}.$$
$$\text{POP } q \Rightarrow \hat{sum} \oplus p \oplus r,$$
$$\Rightarrow \hat{p} \oplus q \oplus r \oplus p \oplus r = \hat{p} \oplus p \oplus q.$$
$$\text{POP } r \Rightarrow \hat{sum} \oplus p \oplus q,$$
$$\Rightarrow \hat{p} \oplus q \oplus r \oplus p \oplus q = \hat{p} \oplus p \oplus r.$$

(We omit the case of disturbing access to q or r, because it causes a similar result).

This shows that a fault injection leads to a fault-infective computation because the faulty *sum* propagates through the subsequent POPs. For this reason, *sum* must be intact for an attacker to succeed. It is also possible to inject faults while accessing primes in the execution of POPs. More specifically, fault injection, making the faulty T_{sum}, outputs the faulty $\hat{sum}$, which results in a faulty infective computation (see line 5 in Algorithm 5). The primes can also be accessed at the CRT recombination. To protect the recombination from fault attacks, we place a security comparison right after the recombination. Thus, if a fault makes a faulty $\hat{p}$, for example, another fault should be injected during the computation of S_{pr} mod [POP p] such that $\hat{S} \not\equiv S_{pr}$ mod [POP p] becomes false, which enables an attacker to pass the security comparison. However, the probability for an attacker to succeed in injecting faults at two precise parts of the computation is negligible [14].

The second type of fault is those that could be injected in a transient manner during any computation. Our algorithm performs a total of three security comparisons. The first is to detect either a faulty S_{pr} or S_{qr}. To that end, $S_{pr} \equiv S_{qr}$ mod [POP r] is checked at line 10 in Algorithm 6. Since we have $\Pr[\hat{S}_{pr} \equiv S_{qr} \bmod r] \approx 2^{-(|r|-1)} \ln 2$, the bit length of r determines the security level of the algorithm. The second is to mainly detect faults that are possibly injected during the computation of $S_p \leftarrow S_{pr}$ mod [POP p] and $S_q \leftarrow S_{pr}$ mod

[POP q] or during the CRT recombination. Since the bit length of p and q is |S|/2, faulty outputs are unlikely to pass this comparison. The last one, is to verify the integrity of *sum*, which might be distorted during the last execution of POP; [POP q]. For this purpose, all primes are newly loaded from the storage and XORed with *sum*; the result must be 0 if all operations have been normally performed without faults. Consistency checks at lines 10 and 16, and the use of POP and PUSH to guarantee the integrity of p, q and r, are closely correlated with each other so that they verify that no error occurred during the computation of S_p or S_q and the CRT recombination.

V. Experimental Setup and Results

In this section, we first describe our fault-injection setup. We then detail the experiment we performed and present the results. We first show the vulnerabilities of Shamir's countermeasure and show the security of our proposed CRT-RSA against fault attacks using the SCARF system.

1. Experimental Setup

For injecting faults and analyzing the response during the execution of CRT-RSA, we need a parameter-controlling PC, an experimental board for fault injection, and an oscilloscope. Specifically, implementing CRT-RSA on a software board is more recommendable than porting on a smart card, because we have to insert an additional trigger for injecting a fault during a specific period. Figure 1 shows the hardware setup that we use to inject faults into an Atmega 128 software board. We implemented both Shamir's countermeasure and our proposed algorithm on an Atmega 128 microcontroller that can operate within the standardized smart card communication specification [19]. The setup consists of a fault-injection control system, named SCARF, our board containing an Atmega 128 microcontroller, a field programmable gate array (FPGA), a triggercontroller, and more. SCARF sends a command including operating parameters to the microcontroller, which in turn forwards the fault parameters to the FPGA. The FPGA then controls the operating environment of the on-board algorithm. The parameters we can handle include the clock cycle and power supply. To inject a fault during the execution of CRT-RSA, we change one of these parameters into abnormal values.

Basically, the parameters consist of timing information and fault types. Timing information specifies an offset, synchronizing duration with the control devices and fault injection duration, as shown in Fig. 2. Fault types can be given by an abnormal clock cycle (Fig. 3), power supply (Fig. 4), or a

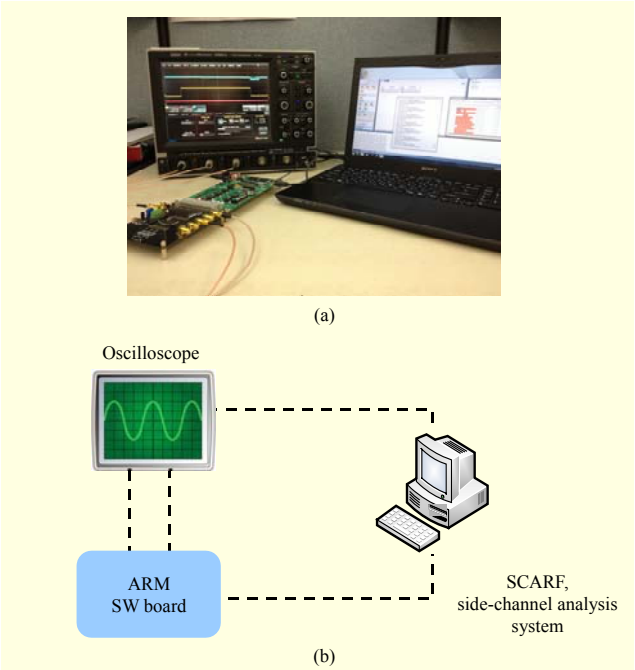


Fig. 1. Fault-injection setup. Atemga 128 microprocessor, connected to PC, forwards fault parameters to FPGA. FPGA then controls operating environment. We can see power traces and clock signals through oscilloscope (a LeCroy WaveRunner 104Mxi-A): (a) experiment setting and (b) setting block diagram.

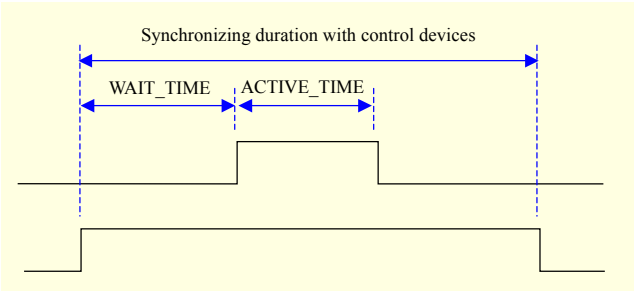


Fig. 2. Parameters for fault injection.

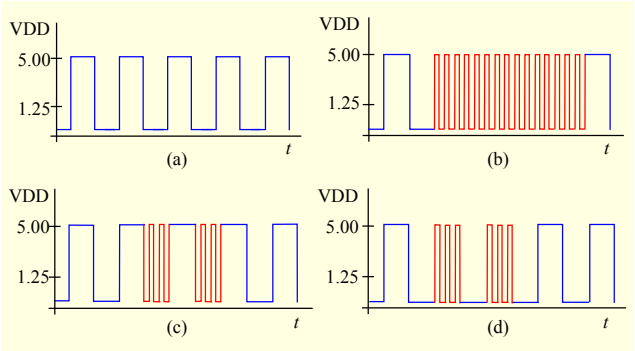


Fig. 3. Clock variances for fault injection: (a) normal case and (b) - (d) clock glitches for fault. Blue line indicates normal clocks, but red line indicates abnormal ones.

ETRI Journal, Volume 36, Number 3, June 2014
http://dx.doi.org/10.4218/etrij.14.0113.0317

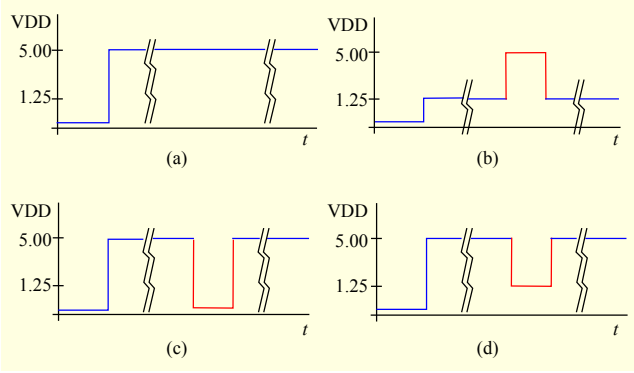


Fig. 4. Variance of power supply for fault injection: (a) normal case and (b) - (d) abnormal power supply for fault.

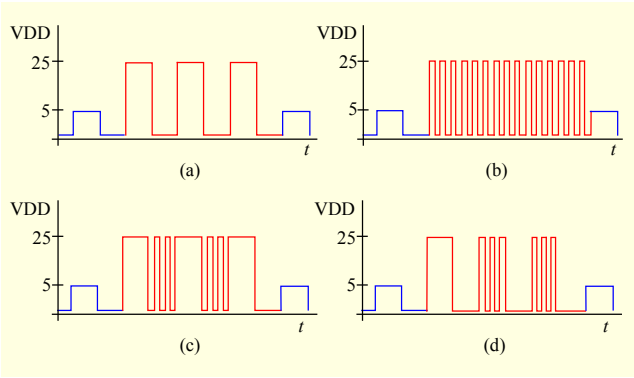


Fig. 5. Combinations of clock glitch and abnormal power supply for fault injection.

combination of the two (Fig. 5). In our experiments, we use an abnormal power supply as a fault.

The Atmega 128 [20] is a high-performance, low-power AVR 8-bit microcontroller, providing an advanced RISC architecture with 133 instructions and up to a 16 MIPS throughput at 16 MHz. It contains 128 KB of in-system reprogrammable flash, 4 KB of EEPROM/internal SRAM, and up to 64 KB of optional external memory space. Its operating voltages are from 4.5 V to 5.5 V. It may not be acceptable for an 8-bit processor to execute CRT-RSA with a modulus size of more than 1,024 bits.

Owing to this fact, we implement both Shamir's countermeasure and our algorithm with $|p| = |q| = 32$ bits, which is reasonable for an Atmega 128. More specifically, the public exponent and secret primes we used are:

- e: 0x03.
- P: 0xD0 67 8A 45.
- Q: 0xE8 09 85 7B.

2. Performed Experiment and Results

Both Shamir's countermeasure and our algorithm are designed to return an error code in the presence of faults. However, if faults are injected without being detected, a meaningful output will be returned; thus, enabling a Bellcore attack. In the following experiments, we intentionally injected faults at the vulnerable points where Shamir's countermeasure is susceptible and injected the same type of faults on the proposed scheme. The results show that Shamir's countermeasure returns a faulty signature, while our algorithm returns a meaningless error code; 0x00.

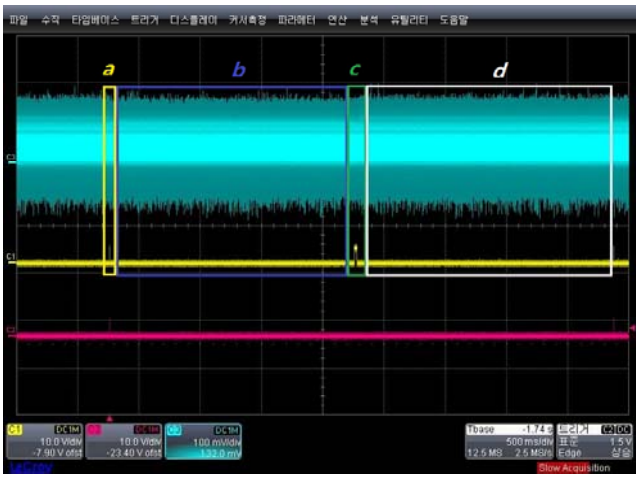
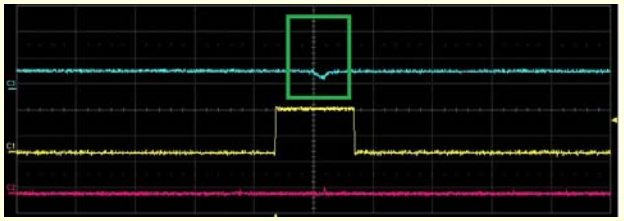


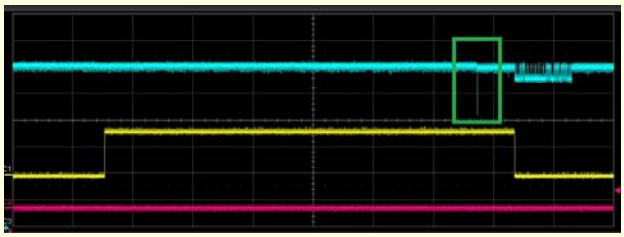
Fig. 6. Waveform of Shamir's countermeasure: blue color is power trace of countermeasure; yellow color indicates trigger signals at rising edge: *a* represents prime loading, *b* represents computation of S_{pr} and S_{qr}, *c* represents $S_p \leftarrow S_{pr} \bmod p$ and $S_q \leftarrow S_{qr} \bmod q$, and *d* represents CRT recombination. Barrett reduction for five modular operations spent most of time. Oscilloscope settings: 500 ms/div and 2.5 MS/s.

Experiment 1 (Shamir's countermeasure): We initiate communication between the Atmega 128 software board and the SCARF system, introduced previously, as defined in the ISO/IEC 7816-4 standard, which specifies the organization, security, and commands for an interchange. Shamir's countermeasure is executed at the command from SCARF, and this command also defines the fault parameters, which include timing information and fault types. It is not possible to extract a trigger signal from a smart card, and for this reason, we used a software board. We set trigger signals generated when particular operations begin so that we can find a part of the specific operation(s) in the waveform of CRT-RSA (Fig. 6). The SCARF system provided us with a user interface controlling the fault-injection timing in the unit of μs. After finding the exact position of the target operations using the trigger signal, we can inject a fault at the right position. We note that a fault-injection environment should support parameters in the unit of μs, and we therefore inject a fault at a fairly lightweight operation such as the loading of a secret prime.

We reduce the 3.3 V power supply to 2.2 V, while (a) the



(a)



(b)

Fig. 7. Fault injection into Shamir's countermeasure. Prime loading and CRT recombination are performed at rising edge in yellow color. Green box shows a power drop. Oscilloscope setting for (a) fault injection at loading of secret primes: 50 MB/s and 5 μs/div and (b) fault injection at CRT recombination: 5 MS/s and 2 ms/div.

									WAIT_TIME [msec]
▶	7	3E	52	17	07	3A	6C	F0	18	20623 [2016,92]
	18	A1	9E	B0	BF	36	1B	70	1C	20623 [2016,92]
	20	00	00	00	00	0E	4B	2B	5C	20623 [2016,92]
	21	3F	87	FD	50	09	C1	97	21	20623 [2016,92]
	24	02	B7	1F	5C	C9	41	94	7D	20623 [2016,92]
	25	00	00	00	00	23	9F	76	4E	20623 [2016,92]
	31	09	87	88	34	1F	60	7A	E0	20623 [2016,92]
	41	B9	C9	B9	76	80	0D	40	DB	20623 [2016,92]
	42	B9	C9	B9	76	80	0D	40	DB	20623 [2016,92]
	44	BA	90	C9	6B	B3	C3	87	EE	20623 [2016,92]

Fig. 8. First ten results of faulty signatures from Shamir's countermeasure. Red-colored bytes are faulty. By using a Bellcore attack and one of these faulty signatures, attacker can reveal the secret primes.

secret primes are being loaded and (b) the operations including the CRT recombination are being performed. We ran procedure (a) 100 times and attack procedure (b) 500 times, trying to inject faults into various points of operations. We set the trigger signals for (a) and (b), and adjusted the timing for a fault injection using the SCARF system, the results of which are shown in Fig. 7. We noticed that a fault injection is not always successful; a software board (or a smart card) often totally crashes or sometimes a fault injection has no effect. As a result, we only succeeded in injecting 60 faults out of a total of 600 ((a) 4 times and (b) 56 times); the faulty outputs are shown

									WAIT_TIME [msec]
▶	11	00	00	00	00	00	00	00	00	20527 [200,..
	33	00	00	00	00	00	00	00	00	20471 [200,..
	34	00	00	00	00	00	00	00	00	20803 [203,..
	66	00	00	00	00	00	00	00	00	20849 [203,..
	100	00	00	00	00	00	00	00	00	20904 [204,..
	101	00	00	00	00	00	00	00	00	20811 [203,..
	110	00	00	00	00	00	00	00	00	20804 [203,..
	156	00	00	00	00	00	00	00	00	20598 [201,..
	177	00	00	00	00	00	00	00	00	20706 [202,..
	221	00	00	00	00	00	00	00	00	20551 [200,..

Fig. 9. First ten return values from our countermeasure. All are pre-defined error code 0x00.

in Fig. 8. The SCARF system first runs the algorithm without faults and stores its original output. Every time faults are injected into the algorithm, the system compares the output of the algorithm to the original output and shows faulty bytes in red if they are different from the original output. By using one of any fault signature shown in Fig. 8, the attacker can reveal the secret primes p and q, and compute then d, because the public exponent e is known.

Experiment 2 (The proposed algorithm): In this experiment, we testify the security of our algorithm. As in Experiment 1, we gave the same type of fault at every point where Shamir's countermeasure fails to protect from faults. For a stronger verification of the security of the proposed algorithm, more faults are injected than had been for Experiment 1. We intensively injected faults when loading secret primes and operations around where the CRT recombination is executed. As pointed out previously, fault injection does not show a fixed success rate even though basic operations are the same. In this experiment, the microcontroller crashed more often than in the previous experiment, and more faults failed to be injected. As a result, only 21 of 1,000 faults are successfully injected and are represented in red in Fig. 9 ((a) 5 times and (b) 16 times). As designed for protection against fault attacks, all faulty signatures return an error code 0x00 without exception. Needless to say, error code 0x00 does not enable a Bellcore attack.

VI. Conclusion

The Bellcore attack on CRT-RSA reveals the secret prime factors by introducing a single fault on a chip. Although Shamir's software countermeasure has attracted a lot of attention, several drawbacks—from a security point of view—

have limited its widespread use. Many later countermeasures have depended on additional exponentiations and inversions to detect fault attacks, resulting in severe performance degradation. In this paper, we improved Shamir's countermeasure to protect against a Bellcore attack with low additional costs. Unlike previous fault injection experiments, our experiment environment, the SCARF system, enables us to inject a fault into the right position, even in the unit of μs. We implemented Shamir's countermeasure, as well as our own countermeasure, on an Atmega 128 software board and generated trigger signals when specific operations began. The exact time of the target operations could be calculated using these trigger signals, and we were able to inject faults wherever we wanted—even at the loading of the secret primes. Our security analysis and experiments demonstrate that the proposed countermeasure provides reliable security.

References

[1] J. Park et al., "Differential Fault Analysis for Round-Reduced AES by Fault Injection," *ETRI J.*, vol. 33, no. 3, June 2011, pp. 434–442.

[2] J. Blomer, M. Otto, and J.-P. Seifert, "A New CRT-RSA Algorithm Secure Against Bellcore Attacks," *Tenth ACM Conf. Comput. Commun. Security*, Washington, DC, USA, Oct. 27–30, 2003, pp. 311–320.

[3] J. Blomer and M. Otto, "Wagner's Attack on a Secure CRT-RSA Algorithm Reconsidered," *Third Int. Conf. Fault Diagnosis Tolerance Cryptography*, Yokohama, Japan, 2006, pp. 13–23.

[4] C. Aumuller et al., "Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures," *Cryptographic Hardware Embedded Syst.*, Redwood Shores, CA, USA, Aug. 13–15, 2002, pp. 260–275.

[5] A. Boscher, H. Handschuh, and E. Trichina, *Fault Resistant RSA Signatures: Chinese Remaindering in Both Directions*. Accessed Mar. 31, 2014. http://eprint.iacr.org/2010/038

[6] D. Vigilant, "RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks," *Tenth Int. Conf. Cryptographic Hardware Embedded Syst.*, Washington, DC, USA, Aug. 10–13, 2008, pp. 130–145.

[7] S.-M. Yen et al., "RSA Speedup with Chinese Remainder Theorem Immune against Hardware Fault Cryptanalysis," *IEEE Trans. Comput.*, vol. 52, no. 4, Apr. 2003, pp. 461–472.

[8] D. Wagner, "Cryptanalysis of a Provably Secure CRT-RSA Algorithm," *Eleventh ACM Conf. Comput. Commun. Security*, Washington, DC, USA, Oct. 25–29, 2004, pp. 92–97.

[9] S.-K. Kim et al., "An Efficient CRT-RSA Algorithm Secure against Power and Fault Attacks," *J. Syst. Software*, vol. 84, no. 10, Oct. 2011, pp. 1660–1669.

[10] S.-M. Yen, D. Kim, and S.J. Moon, "Cryptanalysis of Two Protocols for RSA with CRT Based on Fault Infection," *Third Int. Conf. Fault Diagnosis Tolerance Cryptography*, Yokohama, Japan, vol. 4236, 2006, pp. 53–61.

[11] J.-S. Coron et al., "Fault Attacks and Countermeasures on Vigilant's RSA-CRT Algorithm," *Seventh Int. Conf. Fault Diagnosis Tolerance Cryptography*, Santa Babara, CA, USA, Aug. 21, 2010, pp. 89–96.

[12] D. Boneh, R.A. DeMillo, and R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Advances in Cryptology Sixteenth Annual Int. Conf. Theory Appl. Cryptographic Tech.*, Konstanz, Germany, May 11–15, 1997, pp. 37–51.

[13] A. Shamir, *Improved Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attacks*, US Patent 5,991,415, filed May 12, 1997, issued Nov. 23, 1999.

[14] C. Giraud, "An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis," *IEEE Trans. Comput.*, vol. 55, no. 9, Sept. 2006, pp. 1116–1120.

[15] S.-M. Yen et al., "RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis," *Fourth Int. Conf. Info. Security Cryptology*, Seoul, Rep. of Korea, Dec. 6–7, 2001, pp. 397–413.

[16] S.-K. Kim et al., "An Efficient CRT-RSA Algorithm Secure against Power and Fault Attacks," *J. Syst. Softw.*, vol. 84, no. 10, Oct. 2011, pp. 1660–1669.

[17] A. Boscher, R. Naciri, and E. Prouff, "CRT RSA Algorithm Protected against Fault Attacks," *First Workshop Info. Security Theory Practice*, Crete, Greece, May 9–11, 2007, pp. 229–243.

[18] M. Ciet and M. Joye, "Practical Fault Countermeasures for Chinese Remaindering Based RSA," *Second Int. Conf. Fault Diagnosis Tolerance Cryptography*, Scotland, UK, Sept. 2, 2005, pp. 124–131.

[19] ISO 7816, "Identification Cards Integrated Circuit(s) Cards with Contacts," Geneva, Switzerland, Created in 1989, amended in 1992.

[20] Atmega 128 specification. Accessed Mar. 31, 2014. http://www.atmel.com/Images/doc2467.pdf

Seungkwang Lee received his BS in computer science and electronic engineering from Handong University, Pohang, Rep. of Korea in 2009, and his MS degree in computer science from Pohang University of Science and Technology (POSTECH) Pohang, Rep. of Korea in 2011. He is currently working as a researcher with ETRI, Daejeon, Rep. of Korea. His research interests include side-channel attacks, fault attacks, and software/hardware implementation.

Dooho Choi received his BS degree in mathematics from Sungkyunkwan University, Seoul, Rep. of Korea in 1994 and his MS and PhD in mathematics from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea in 1996 and 2002, respectively. Since Jan. 2002, he has been a senior researcher in Electronics and Telecommunications Research Institute (ETRI), Daejeon, Rep. of Korea. His current research interests are side-channel analysis and its resistant crypto design, security technologies of RFID and wireless sensor networks, lightweight cryptographic protocol/module design, and cryptography based on non-commutativity. He was the editor of the ITU-T Rec. X.1171.

Yongje Choi received his BSEE and MS from Chonnam National University, Gwangju, Rep. of Korea, in 1996 and 1999, respectively. He is currently a senior member of technical staff at the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Rep. of Korea. His research interests include VLSI design, crypto processor design, side-channel analysis, and information security.

http://dx.doi.org/10.4218/etrij.14.0113.0317