Journal of Computer and Communications, 2014, 2, 1-9
Published Online March 2014 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2014.24001

Scientific Research

Timing Attack Analysis on AA_β Cryptosystem

A. H. A. Ghafar[1], M. R. K. Ariffin[2]

[1]Al-Kindi Cryptography Research Laboratory, Institute for Mathematical Research, Universiti Putra Malaysia (UPM), Serdang, Malaysia
[2]Mathematics Department, Faculty of Science, Universiti Putra Malaysia (UPM), Serdang, Malaysia
Email: amirghafar87@gmail.com, rezal@upm.edu.my

Received September 2013

Abstract

Timing attack is an attack on the implementation of a cryptographic primitive. The attack collects leaked secret data via certain implementation techniques either on software or hardware. This paper provides an analysis of a theoretical timing attack on the AA_β algorithm. The attack discussed in this paper gives avenues for secure implementation of AA_β against timing attacks. The simulation of the attack is important to provide invulnerability features for the algorithm in order to be implemented and embedded on applications. At the end of the attack, a method to overcome it will be introduced and it is called AA_β blinding.

Keywords

Timing Attack; Side-Channel Attack; Public-Key Cryptosystem; AA_β Public Key Cryptosystem

1. Introduction

The AA_β cryptosystem utilizes the integer factorization problem (IFP) coupled with the square root problem as its cryptographic primitive. The algorithm claims to have a complexity order of $O(n^2)$ compared to renowned RSA and ECC algorithms which both have $O(n^3)$ in encryption and decryption process [1]. Although Rabin cryptosystem also shares the same cryptographic primitive, but its decryption is 4-to-1 which makes the decrypted plaintext hard to distinguish. But AA_β overcomes this decryption failure without losing computational power and speed.

Several cryptanalysis have been conducted on this cryptosystem and have pointed out that to find solution for the cryptosystem is infeasible [1]. However, this cryptosystem has never been analyzed or attacked from implementation point of view. The reason for this is because AA_β has not yet been implemented in a real-world application. Nevertheless, it is important to launch a theoretical attack on its implementation to provide future guidance for future implementers. This paper intends to do that and we will use one common type of side channel attack which is timing attack.

Timing attack uses leaked information of a cryptographic implementation to derive the secret keys of cryptographic primitives. The first timing attack was developed to recover secret key, d in RSA implementation of smart card [2]. Consider we have decryption equation of RSA $M \equiv C^d \pmod N$ for C is n-bit number and d is k-bit number. By measuring the computational time of the RSA decryption, the attack is able to collect the private key bit per bit. In many cases, the attacker has unlimited access to the decryption machine but does

How to cite this paper: Ghafar, A.H.A. and Ariffin, M.R.K. (2014) Timing Attack Analysis on AA_β Cryptosystem. *Journal of Computer and Communications*, **2**, 1-9. http://dx.doi.org/10.4236/jcc.2014.24001

not have any knowledge about the secret key.

In order to prevent this kind of attack on RSA, Rivest introduced a defense mechanism called RSA blinding [3]. It works by generating a random number, r which will 'blind' the product of $C^d \pmod N$ and essentially makes it impossible for attacker to determine the computational time. We will make use of this blinding concept to come out with our own strategy for the AA_β algorithm. Like the RSA, random numbers also play a role in our blinding mechanism to make sure that the decryption process time will be obscured.

This paper is structured as follows. Section 2 will dive into some algorithms that are commonly used to optimize both modular multiplication and exponentiation. Section 3 gives an overview of AA_β algorithm while we will discuss on how timing attack is to be conducted on the algorithm in Section 4. We will also discuss the AA_β blinding in Section 5 to prevent any further timing attack on it in the future. Future works and conclusion are in Section 6 and 7 respectively.

2. Design of Timing Attacks on Modular Exponentiation and Multiplication

The process of modular exponentiation and modular multiplication is essentially very time consuming process especially when involves exponentiation of very large integers. They can slow down the whole process of decryption and this is non-economical in commercial applications such as smart cards and real-time web communication which need fast response time. However, various methods have been introduced to speed up both operations.

First and the most basic method to increase the computational speed of this equation is repeated squaring algorithm. This algorithm is mostly used in power constrained hardware such as smart card due to its low complexity in calculation. However, it was exploited by Kotcher in the earliest timing attack on cryptosystem's hardware. Consider the decryption process of RSA.

$$P \equiv C^d \pmod N$$

Kotcher cleverly observes this process as signal which has been corrupted by the 'noise' of unknown bits of d. The attack emulates the decryption process and captures the calculated time of decryption using chosen private keys until each bits of actual d is exposed.

Generally, for chosen private keys, u and v which have size of n-bits

$$u = u_0, u_1, \ldots, u_{i-1}, 0, u_{i+1}, \ldots u_{n-1}, u_n$$

$$v = v_0, v_1, \ldots, v_{i-1}, 0, v_{i+1}, \ldots v_{n-1}, v_n$$

the only difference between both chosen private keys is bit at point i (Assume that both u and v already have same arrangement of bits with actual private key, d up until position i−1).

Let C_j be the calculation time to complete decryption using the actual private key, d. Also let T_u be the calculation time from bit u_0 to u_i using first emulated key, u and T_v be the calculation time from bit v_0 to v_i using second emulated key, v. We assign

$$\left(T'_{u,v}\right) = C_j - T_{u,v}$$

Hence, variance of timing after point i for both u and v are regarded as

$$\mathrm{var}\left(C_j - T_u\right) = \mathrm{var}\left(T'_u\right) + \mathrm{var}(T_e)$$

and

$$\mathrm{var}\left(C_j - T_v\right) = \mathrm{var}\left(T'_v\right) + \mathrm{var}\left(T_e\right)$$

where T_e is the measurement error. If we assume u has the correct bit (*i.e.* d has bit 0 at point i), then

$$\mathrm{var}\left(T'_u\right) < \mathrm{var}\left(T'_v\right)$$

due to increased similarities in u in terms of bits compared to v.

However, this kind of attack is only successful if the implementation uses repeated squaring algorithm. In more advanced RSA implementations, repeated squaring algorithm is complemented with Montgomery reduction algorithm for every multiplication operation. Montgomery reduction algorithm, skips modular reduction

which uses the 'expensive' division operation. Montgomery reduction transforms the multiplication of

$$a \cdot b (\text{mod } N)$$

to

$$a' = aR (\text{mod } N) \text{ and } b' = bR (\text{mod } N)$$

for which we said a' and b' are in Montgomery form and $R = 2^k$ for $k \in \mathbb{Z}$ [4]. The algorithm executes multiplication in R reduction where a computer can easily and cheaply calculate because the division of 2^k in binary is just a simple shift by k bits. One good explanation and example of Montgomery reduction can be read here [5]. The pseudo codes of both algorithms and how they work together are explained in **Algorithm 1**.

To calculate modular exponentiation, most implementations use the Chinese Remainder Theorem (CRT) approach. This approach divides the modular exponentiation which is in the form of modular $N = p \cdot q$ to the form of modulo p and q. Values of p and q are normally stored in decryption machine. The logic behind this approach is because the value of p and q are smaller than N, hence the modular exponentiation will be more computationally efficient due to their decreased size. To get the original result, the results in modulo p and modulo q are combined.

Later, Schindler produced a significant observation for RSA implementation that uses repeated squaring, CRT and Montgomery reduction approach [6]. Schindler observes that the extra reduction in Montgomery algorithm to ensure $s \leq N$ (refer **Algorithm 2**) caused atiming difference for different a and b. From this observation, Schindler derived the probability of an extra reduction occurring for $a_p^d (\text{mod } p)$ which is

$$P(\text{extra reduction}) = \frac{a(\text{mod } p)}{2R} \tag{1}$$

Observation from (1) shows that there is discontinuity after certain number of extra reduction process which happens to be a multiple of p.

When a is approached by values less than a multiple of p as shown in **Figure 1**, number of reduction

Algorithm 1. Repeated squaring algorithm.

```
//Find y = x^d (mod N)
//where d = (d_0, d_1, d_2, ... d_{n-1}) with d_0 = 1
t' = x · R (mod N)
s' = t'
for i = 1 to n - 1
    s' = Montgomery(s', s')
    if d_i == 1 then
        s' = Montgomery(s', t')
    end if
next i
t' = s' · r' (mod N)
return (t)
```

Algorithm 2. Montgomery algorithm.

```
//Find a' · b'
//Where a' = a · R (mod N) and b' = a · R (mod N)
//Given R · R' - N · N' = 1
Montgomery(a' · b')
    z = a' · b'
    r = (z (mod R))(N' (mod R))
    s = (z + r · N) = R
    if s ≥ N then
        s = s - N //extra reduction process
    end if
    return (s)
end Montgomery
```

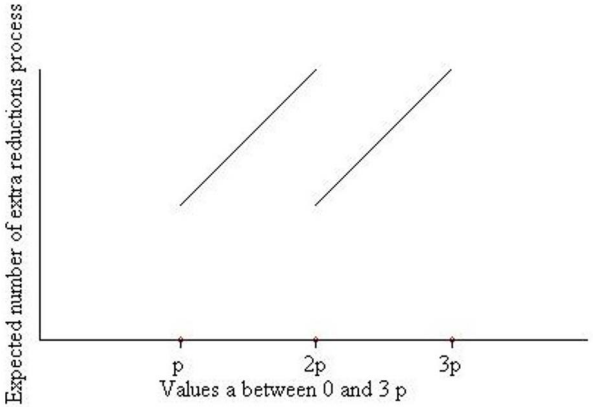

Figure 1. Number of extra reductions in a Montgomery reduction as function (1) of the input *a*.

increases. But, right after the multiple of p, the number of extra reduction drops significantly. This drop can help us to derive the multiple of p hence factorize N.

Another algorithm known as the Karatsuba multiplication is regarded as the most efficient multiplication algorithm of two numbers with the same numbers of digits [7]. This works by assuming that addition is much cheaper than multiplication operation. The algorithm is explained as follows.

Given that we want to multiply a and b that have n numbers of digits. We can write

$$a = a_0 + a_1 \cdot 10^{n/2-1}$$

$$b = b_0 + b_1 \cdot 10^{n/2-1}$$

Basic arithmetic shows that

$$\begin{aligned} a \cdot b &= (a_0 + a_1 \cdot 10^{n/2-1})(b_0 + b_1 \cdot 10^{n/2-1}) \\ &= a_0 b_0 + a_0 b_1 \cdot 10^{n/2-1} + a_1 b_0 \cdot 10^{n/2-1} + a_1 b_1 \cdot 10^{2(n/2-1)} \\ &= a_0 b_0 + 10^{n/2-1}\left(a_0 b_1 + a_1 b_0\right) + a_1 b_1 \cdot 10^{2(n/2-1)} \end{aligned}$$

If we define

$$z_0 = a_0 b_0$$

$$z_1 = a_0 b_1 + a_1 b_0$$

$$z_2 = a_1 b_1$$

It is easy to see that

$$z_1 = \left(a_0 - a_1\right)\left(b_0 - b_1\right) - z_0 - z_2$$

And the product of a and b is

$$a \cdot b = z_0 + \left(z_1 \cdot 10^{n/2-1}\right) + \left(z_2 \cdot 10^{2(n/2-1)}\right).$$

Karatsuba algorithm cuts the work factor $O(n^2)$ in long normal multiplication to only $O(n^{1.585})$.

Both Montgomery multiplication and Karatsuba algorithm have been extensively used in real implementation of RSA. RSA implementation in OpenSSL used repeated squaring, Montgomery multiplication and sliding windows while utilized Karatsuba when multiplying a and b that have same size.

There are two vulnerabilities which have been exploited in this attack. First is the extra reduction happened in Montgomery algorithm. This has been explained in Schindler's attack while the other is the usage of long multiplication and Karatsuba algorithm. Both multiplications show different time significantly [8]. Brumley and Boneh showed how a timing attack launched on RSA implementation in OpenSSL [9]. The attack is so elegant, it was done remotely between two servers in two different buildings. Every time the Mongomery form of C

slightly moves toward less than p, the number of extra reductions in the Montgomery reduction (**Algorithm 2**) increases. It involves the multiplication of two numbers that are about the same size hence Karatsuba multiplication is used. Meanwhile, whenever the value of C moves to be larger than p, the re- duction in **Algorithm 2** will decrease. Long multiplication is used here because the multiplication involves two numbers that have significant different sizes.

The attack's aim is to find the binary structure of p:

$$p = (b_0, b_1, \ldots, b_{i-1}, b_i, \ldots, b_{n-1}, b_n)_2$$

where $b_0 = 1$ and has size of n-bits. This will solve the factorization of $N = p \cdot q$ and lead to a total break of RSA algorithm. The following method is used to recover the bit of p at point-i:

1) Suppose that bits of p have been determined from b_0 to b_{i-1}. Let

$$C_0 = (b_0, b_1, \ldots, b_{i-1}, 0, \ldots 0, 0)$$

and

$$C_1 = (b_0, b_1, \ldots, b_{i-1}, 1, \ldots 0, 0)$$

That is, the only difference between the two chosen ciphertexts is the bit at point-i. C_0 and C_1 are the candidates of prime p.

2) Assign $T(C_0)$ and $T(C_1)$ as the measured decryption times for C_0 and C_1 respectively.

3) Assign $\Delta = \left| T(C_0) - T(C_1) \right|$ *i.e* the timing difference between $T(C_0)$ and $T(C_1)$.

4) If $C_0 < p < C_1$, then Δ is "large". This indicates $b_i = 0$. If $C_0 < C_1 < p$, then Δ is "small". This indicates $b_i = 1$. The thresholds of "large" and "small" are set by the previous values of Δ.

5) The previous steps are repeated to obtain bits at points $b_{i+1}, b_{i+2}, b_{i+3}, \ldots$ until half of the bits p have been recovered.

After obtaining half of bits of p, Coppersmith has extensive methods to recover the whole bits of p and consequently factorizes N [10]. Value of $\triangle$ reveals whether extra reductions of Montgomery multiplication or multiplication (normal versus Karatsuba) predominates during the decryption process at point i.

3. AA_β Algorithm

The Rabin cryptosystem utilized the square root modulo problem as its cryptographic primitive [11]. The alluring factor in Rabin cryptosystem is it has a very small exponent key, $e = 2$. But the cryptosystem has a decryption failure which is caused by the 4-to-1 mapping during decryption. This trade-off deterred Rabin cryptosystem to be widely used in modern application.

In 2013, Ariffin *et al*. proposed a new cryptosystem which also utilized the same problem but without any decryption failure. The algorithm is workable with lower complexity order compared to DHKE, El-Gammal, RSA and ECC [1]. **Algorithms 3-5** show the whole cryptosystem.

The decryption algorithm uses multiplication together with a one time modular reduction. This lack of expensive operation makes AA_β to be an ideal candidate for future public key cryptosystem.

We now analyze the operation given by

$$W \equiv C \cdot d \pmod{pq}$$

Algorithm 3. AA_β key generation.

repeat
  Pick random primes p and q of bit-size n,
until $p \equiv 3 \pmod 4$ and $q \equiv 3 \pmod 4$.
Pick a random integer such that $d > (p^2 q)^{\frac{4}{9}}$.
Calculate integer e, such that $ed \equiv 1 \pmod{pq}$
repeat
  Add multiples of pq to e
until $2^{3n+4} < e < 2^{3n+6}$
Compute pq and $e_{A1} = p^2 q$.
Assign $e_{A2} = e$.
Return (n, e_{A1}, e_{A2}) as public parameters and (pq, d).

Algorithm 4. AA_β encryption.

Read plaintext, m in binary form.
Divide m to m_1 (be a $3n+1$ bit integer)
and m_2 (be a $n-1$ bit integer).
Pick a random integer k_1.
Compute $U = (m_1 \cdot 2^n) + k_1$.
Pick a random integer k_2.
Compute $V = (m_2 \cdot 2^n) + k_2$.
Compute $C = U \cdot e_{A1} + V^2 \cdot e_{A2}$.
Return ciphertext, C.

Algorithm 5. AA_β decryption.

Read ciphertext, C in binary form.
Compute $W \equiv C \cdot d \pmod{pq}$.
Compute $M_1 \equiv q^{-1} \pmod{p}$.
Compute $M_2 \equiv p^{-1} \pmod{q}$.
Compute $x_p \equiv W^{\frac{p+1}{4}} \pmod{p}$
Compute $x_q \equiv W^{\frac{q+1}{4}} \pmod{q}$
Compute $V_1 \equiv x_p M_1 q + x_q M_2\, p \pmod{pq}$.
Compute $V_2 \equiv x_p M_1 q - x_q M_2\, p \pmod{pq}$.
Compute $V_3 \equiv -x_p M_1 q + x_q M_2\, p \pmod{pq}$.
Compute $V_4 \equiv -x_p M_1 q - x_q M_2\, p \pmod{pq}$.
for i from 1 to 4 **do**

$$U_i = \frac{C - V_i^2 \cdot e_{A2}}{e_{A1}}$$

end **do**.
repeat
Check pair (U_i, V_i)
until (U_i, V_i) = integer.
Compute $m_1 = \left\lfloor \frac{U_i}{2^n} \right\rfloor$
Compute $m_2 = \left\lfloor \frac{V_i}{2^n} \right\rfloor$
Compute $m = m_1 \cdot 2^n + m_2$.
Return plaintext, m.

and discuss whether the time calculation of this operation can be exploited. Later, we will enhance the capability of an attacker to obtain W which consequently will continue onto the calculation of

$$x_p \equiv W^{\frac{p+1}{4}} \pmod{p}$$

and

$$x_q \equiv W^{\frac{q+1}{4}} \pmod{q}$$

assuming that we can distinguish the operational time for all multiplications that occurs within the decryption machine. In the next section, we will show how the attack is launched on the AA_β decryption algorithm.

4. Timing Attack on AA_β Algorithm

From Section 2, it is obvious that timing attack manipulates the timing difference in exponentiation multiplication with the aim to find first, the bits of private key bit per bit and second, factor $N = p \cdot q$ (*i.e.* for a public key cryptosystem which accommodates hard factorization problem as its underlying strength). We have also observed the multiple stages of the decryption algorithm of AA_β (**Algorithm 5**) in Section 3. Despite its intricacy, the algorithm is still faster than RSA. Since this paper only discusses the theoretical analysis of timing attack due to lack of real-world implementation of AA_β algorithm, it is reasonable to assume that future implementation of this algorithm will adhere to the vastly used implementation in terms of multiplication and

modular exponentiation. Based on this assumption, we now observe the first multiplication in the AA_β decryption algorithm which is

$$W \equiv C \cdot d \pmod{pq} \tag{2}$$

Since $C \sim 2^{7n}$ and $d \sim 2^{2n}$, the inequality in the size of C and d will not give us a choice to use Karatsuba reduction in the implementations of AA_β. This is because Karatsuba algorithm requires both multiplication elements to be in the same size to make it efficient as stated in Section 2.

As for other method, Montgomery multiplication, it is also not practical in this multiplication process because the reduction only works better than long multiplication if there is an exponentiation operation involved. However Equation (2) does not involve exponentiation.

In short, to compute $C \cdot d$ efficiently, one only use long multiplication. Next, we increase the advantage for the attacker by assuming that value W can be manipulated or collected. This loose assumption is realistic due to misuse implementation or hardware faulty. We also assume that equation

$$\mathrm{x_p} \equiv W^{\frac{\mathrm{p}+1}{4}} \pmod{\mathrm{p}} \tag{3}$$

is implemented using the same method of RSA SSL implementation. Implementors are lured to the same method because the method can use dynamically Montgomery and Karatsuba multiplication for fast modular exponentiation—depends on the exponents given.

If that is the case, we will show that p can be obtained by measuring computational time of (3), using the same method by Brumley and Boneh which eventually gives out p. The method is as follows:

1) Suppose that bits of p have been determined from b_0 to b_{i-1}. Let

$$W_0 = (b_0, b_1, \ldots, b_{i-1}, 0, \ldots 0, 0)$$

and

$$W_1 = (b_0, b_1, \ldots, b_{i-1}, 1, \ldots 0, 0)$$

that is, the only difference between the two chosen ciphertexts is the bit at point-i. W_0 and W_1 are the candidates of prime p.

2) Assign $T(W_0)$ and $T(W_1)$ as the measured decryption times for W_0 and W_1 respectively.

3) Assign $\Delta = \left| T(W_0) - T(W_1) \right|$ *i.e.* the timing difference between $T(W_0)$ and $T(W_1)$.

4) If $W_0 < p < W_1$, then Δ is "large". This indicates $b_i = 0$. If $W_0 < W_1 < p$, then Δ is "small". This indicates $b_i = 1$. The thresholds of "large" and "small" are set by the previous values of Δ.

5) The previous steps are repeated to obtain bits at points $b_{i+1}, b_{i+2}, b_{i+3}, \ldots$ until half of the bits p have been recovered.

It is clear that AA_β is susceptible to timing attack if value of W is leaked. Hence further improvement is needed to overcome the attack. Section 5 will show how this kind of attack can be addressed by a method called AA_β blinding.

5. AA_β Blinding

We begin this section under strong assumption that a timing analysis can be conducted upon $C \cdot d$ prior to compute $\mathrm{W} \equiv C \cdot d \pmod{pq}$ and an attacker would be able to derive the private key d bit per bit. Hence there is a need to blind the multiplication process of $C \cdot d$ before the modulo reduction of pq. We called this process as AA_β blinding.

Based on its name, it is clear that the method uses the same principle with RSA blinding.We will introduce a random number, r into AA_β decryption algorithm. Instead of deterministic decryption of time, the algorithm will have randomized timing decryption. The randomization will increase the difficulty for the adversary to find out the exact value of the private key, d. The alteration occurs within the AA_β decryption algorithm (**Algorithm 5**); starting from Step 1. The procedure is detailed in **Algorithm 6**.

Observe

$$W_1 \equiv C \cdot (r \cdot Q) \pmod{pq}$$

and

Algorithm 6. AA_{β} blinding.

Choose random r such that $2^{n-1} < r < 2^{2n}$.
Get Q and R such that $\frac{d}{r} = Q + \frac{R}{r}$

$$W_2 \equiv C \cdot R \pmod{pq}.$$

This value will be computed in

$$x_p \equiv (W_1 + W_2)^{\frac{p+1}{4}} \pmod{p}$$

and

$$x_q \equiv (W_1 + W_2)^{\frac{q+1}{4}} \pmod{q}.$$

It is easy to see that

$$(W_1 + W_2)^{\frac{p+1}{4}} \equiv (C \cdot r \cdot Q + C \cdot R)^{\frac{p+1}{4}} \pmod{p}$$
$$\equiv (C(r \cdot Q + R))^{\frac{p+1}{4}} \pmod{p} \equiv (C \cdot d)^{\frac{p+1}{4}} \pmod{p}.$$

and

$$(W_1 + W_2)^{\frac{q+1}{4}} \equiv (C \cdot r \cdot Q + C \cdot R)^{\frac{q+1}{4}} \pmod{q}$$
$$\equiv (C(r \cdot Q + R))^{\frac{q+1}{4}} \pmod{q} \equiv (C \cdot d)^{\frac{q+1}{4}} \pmod{q}.$$

which are exactly following the original steps 5 and 6 in **Algorithm 5**.

By using this blinding method, we change the deterministic characteristic in the decryption algorithm to a probabilistic one. The size of randomized element, r is n-bit which makes it infeasible to be determined. We will now simulate an example.

Example Let $n = 16$. Along will choose primes, $p = 53887$, $q = 53891$ and $N = 4070567597$. We choose $d = 6381159375677$ and $r = 544373306496$ randomly.

We obtain $Q = 1004100545$ dna $R = 21477$.

The public keys are

1) $e_{A1} = 156489158370179$
2) $e_{A2} = 12180361837718376$

Let the derivatives of the messages that Busu want to send are

1) $U = 9414048941460394886$
2) $V = 1281821555$

Thus, $C = 21486341075284587745963757517831994$. The blinding starts when

$$W_1 \equiv C \cdot r \cdot Q \bmod 4070567597 \equiv 1344015274 \bmod 4070567597$$
$$W_2 \equiv C \cdot R \bmod pq \equiv 2175088081 \bmod 4070567597$$

and Along obtains

$$x_p \equiv (134401527 + 2175088081)^{\frac{53888}{4}} \bmod 53887 \equiv 11486 \bmod 53887$$

$$x_q \equiv (1344015274 + 2175088081)^{\frac{53892}{4}} \bmod 53891 \equiv 24120 \bmod 53891$$

Further calculation based on **Algorithm 5** successfully get back $V_1 = 1281821555$ and consequently $U_1 = 9414048941460394886$.

6. Future Works

Many aspects of side-channel attacks have not been tested yet on the AA_{β} cryptosystems. Other types of attacks are the differential power analysis attack and the glitching attack because it follows the same methodolgy

with the timing attack that has been discussed in this paper.

Furthermore, a deeper analysis is required to test the randomness generated in AA_{β} blinding to make sure that the decyrption algorithm will always be a probabilistic process.

7. Conclusion

AA_{β} decryption algorithm is susceptible to timing attacks that simulates the Brumley and Boneh attack on SSL. However, a method to randomize the output on AA_{β} can overcome this problem. The blinding process changes the deterministic characteristic in the decryption algorithm to become probabilistic and this makes the data sampling procedure for the attack infeasible.

References

[1] Ariffin, M.R.K., *et al*. (2013) A New Efficient Asymmetric Cryptosystem Based on the Square Root Problem. *Malaysian Journal of Mathematical Sciences*, **7**, 19-37.

[2] Kocher, P. (1996) Timing Attacks on Implementations of Difie-Hellman, RSA, DSS, and Other Systems. *CRYPTO'*96, Santa Barbara, 18-22 August 1996, 104-113.

[3] Kaliski, B. (1996) Timing Attacks on Cryptosystems. *RSA Laboratories Bulletin*, **2**.

[4] Montgomery, P. (1985) Modular Multiplication with Trial Division. *Mathematics of Computation*, *Mathematics of Computation*, **44**, 519-521. http://dx.doi.org/10.1090/S0025-5718-1985-0777282-X

[5] Buell, D.A. (2005) Montgomery Multiplication. http://cse.sc.edu/buell/csce557/Dlecturenotes/

[6] Schindler, W. (2000) A Timing Attack against RSA with the Chinese Remainder Theorem. *Workshop on Cryptographic Hardware and Embedded Systems* 2000 (*CHES* 2000), Worcester, 17-18 August 2000, 109-124.

[7] Karatsuba, A. and Ofman, Y. (1963) Multiplication of Many-Digital Numbers by Automatic Computers. *Physics-Doklady*, **7**, 595-596.

[8] Stamp, M. and Low, R.M. (2004) Applied Cryptanalysis. John Wiley & Sons Inc., Hoboken.

[9] Brumley, D. and Boneh, D. (2003) Remote Timing Attacks Are Practical. *Proceedings of* 12*th Conference on USENIX Security Symposium*, 4-8 August 2003, Washington DC.

[10] Coppersmith, D. (1997) Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *Journal of Cryptology*, **10**, 233-260. http://dx.doi.org/10.1007/s001459900030

[11] Rabin, M.O. (1979) Digitalized Signatures and Public Key Functions as Intractable as Factorization. MIT Laboratory for Computer Science, MIT/LCS/TR-212.