Journal of VLSI Design Tools & Technology
ISSN: 2249-474X (Online), ISSN: 2321-6492 (Print)
Volume 7, Issue 3
www.stmjournals.com

Implementation of RSA and CRT-RSA with MIST to Resist Power Analysis Attacks

Hridoy Jyoti Mahanta*, ***Ajoy Kumar Khan***
Department of Computer Science and Engineering, Assam University, Silchar, Assam, India

Abstract
Security has transformed out into a fundamental range of research in space of computer science. With the advancement of side channel attacks, all the private and public key cryptosystems accessible worldwide has been discovered defenceless. Accordingly there has broughtdireness up in planning resisting systems against such unpredictable attacks. Power analysis attacks, which is the most famous side channel attacks has turned out to be a test for the majority of the normal cryptosystems like advanced encryption standard (AES), data encryption standard (DES), Rivets-Shamir-Adleman (RSA), ECC and so forth. MIST is an algorithm which was intended for processing exponentiation. As the most critical operation of RSA is the modular exponentiation for encryption and decryption, MIST can assume an imperative part in planning modified RSA to counteract power analysis attacks. We have extended our work CRT-RSA (Chinese Remainder Theorem–Rivets-Shamir-Adleman) which is more widely used in computation. The analysis results shows that modified RSA can perform better in computing RSAespecially when CRT is has been used. The results have also been verified using VHDL (very high speed integrated circuits).

Keywords: *Power analysis attacks, cryptosystems, RSA, RSA-CRT, MIST, VHDL*

****Author for Correspondence*** E mail: hridoy69@gmail.com

INTRODUCTION

Security has been the most important component in technological scenario as it is responsible for securing all information passed through networked computers. Cryptographic algorithm has been widely used in network computers for secure transactions of secret information. In any case, regardless of these different security techniques presented by various cryptosystems, attacks always have been effective to discover a loophole to break the security of these systems. Side-channel attacks (SCA) are firmly identified with the presence of physically noticeable changes caused by the execution of operations in show microelectronic devices. Side channel attacks are those attacks that depend on side channel information. Side channels Information are those data that can be recovered from the microelectronic devices with no intrusion into the device. Presently the cryptographic devices are under the potential threat of side channel attacks.

Side channel attacks are effective techniques to recoup delicate information of cryptographic gadgets. In cryptography, a side-channel attack is a sort of attack that in view of information picked up from the physical execution of a cryptosystem, as opposed to savage power or hypothetical shortcomings in the algorithms. For case, timing information, control utilization, electromagnetic releases or even stable can give an additional wellspring of information, which can be misused to break the framework some side channel attacks require specialized learning of the inward operation of the framework. Attackers use some or all of these side channel information along withother cryptanalytic techniques to reveal the secret key that usingby the device. Side channel analysis techniques are of high concern because these attacks can be mounted quickly.

Power analysis attacks have been exhibited as a standout amongst the most intense attacks for most direct execution of symmetric and asymmetric ciphers. Power analysis attacks depend on the examination of power utilization of a cryptographic device. Execution of cryptographic operations devours control, these power utilization goes about as side channel information. Power utilization of a device relies upon information it procedures and operations it performs. Cryptographic algorithms, for example, advanced encryption standard (AES), data encryption standard (DES), Rivets-Shamir-Adleman (RSA) and so forth is actualized in cryptographic devices. Execution these algorithms in the devices consumes power, this power consumption acts as side channel information. Attackers trace power consumption of a device on an oscilloscope and analyses the power traces to find out the secrete key of a device. Power analysis attacks are further divided into simple power analysis (SPA) and differential power analysis (DPA) respectively [1]. SPA uses few traces of power consumption to reveal and the cryptosystem and few key bits, whereas DPA uses a large number of power traces and make statistical analysis of them to reveal all the key bits by correlating them with some hypothetical power traces.

DPA attacks worked in light of the fact that the power utilization of cryptographic device was reliant on the halfway estimations of the executed cryptographic algorithms. The objective of countermeasures is evacuating this reliance. The main idea of countermeasures against DPA attacks is to make the power consumption of the cryptographic device independent of the intermediate values of the executed cryptographic algorithm [1]. The countermeasures against DPA attacks that have been distributed so far can basically be classified into two gatherings i.e. randomization and masking. The fundamental thought of stowing away is to expel the information reliance of the power consumption, whereas concept of masking is to randomize the intermediate value that is proceed by cryptographic device.

In this paper, we have used the MIST algorithm which computes modular exponentiation using some random number, with RSA and CRT-RSA to examine the effects on resisting power analysis attacks. The rest of the paper has been organized in following sections. Section 2 gives a detail on the background of how power analysis is resisted with some related works. Section 3 presents RSA and CRT-RSA in details. The MIST algorithm has been briefly discussed followed by our proposed work and lastly shows the result and finally a conclusion has been formulated.

BACKGROUND AND RELATED WORKS

The main reason behind power analysis attacks is that almost all the cryptographic devices are built with CMOS. The CMOS has a special property that it power consumption depends on the operations it performed as well as the data on which they were performed. Countermeasure expels the information reliance of the power consumption. This implies either the execution of the algorithm is randomized or the power consumption qualities of the device are changed such that an aggressor can't without much of a stretch discover information reliance. Power analysis attacks works in light of the fact that the power utilization of cryptographic devices relies upon middle estimations of the executed cryptographic calculations. The objective of countermeasures is to make the power utilization of cryptographic devices free of the intermediate values and autonomous of the operations that are performed [1]. There are basically two ways to deal with accomplish this autonomy. The main approach is to fabricate devices such that the power consumption is arbitrary. This implies in each clock cycle arbitrary measure of energy is devoured. The second approach is to build devices that consume an equal amount of power for all operations and for all data values. Hence, equal amounts of power are consumed in each clock cycle.

Masking could be achieved by randomizing the intermediate values that are processed by the cryptographic device. An advantage of this approach is that it can be implemented at the algorithm level without changing the power consumption characteristics of the cryptographic device. At the end of the day, masking permits make the power consumption autonomous of the middle of the road esteems, regardless of the possibility that the device has information subordinate power consumption [1].

The least complex approach to conceal variety is to make the calculation entirely steady time, for all conceivable mystery examples or stowing away inward state, so that the attacker can't reproduce inner calculations any more. This is finished by breaking the connection between the power consumption of the devices and the prepared information esteems. Consequently, cryptographic devices that are ensured by stowing away execute cryptographic calculations similarly as unprotected devices. In specific, they compute a similar intermediate values. However, the concealing countermeasures make it troublesome for an attacker to discover exploitable data in control traces [1].

The most broad strategy to counter SCA attacks is to randomize information that may spill through different side channels, for example, power consumption, electromagnetic radiation, or execution time. The issue is to ensure that an assailant may acquire just irregular data, and in this way can't increase any helpful learning about the genuine beginning or potentially intermediate information associated with computations. In instance of elliptic curve cryptosystem, randomized projective directions technique is a down to earth countermeasure against SCA attacks in which an attacker can't anticipate the presence of a particular esteem on the grounds that the directions have been randomized [1].

Many works on resisting power analysis attacks on RSA appears in literature. All these works are based on the approaches discussed below. In 2004 Mamiya *et al.* [3] presented the concepts of using random initial points in computations which could start computations are at random points every time. This could actually randomize the operations in any computations. In 2005, Kim *et al.* [4] presented an approach that could immune RSA from power analysis attacks. In their computation they used a random number which was first multiplied and then the inverse of was computed to remove the effect of the random number. They also proposed an improve countermeasure against SPA and DPA in order to avoid reverse computation of modular exponentiation [5]. It is not necessary to compute an inverse of the random number r. It is very important to speed up the RSA computation and implementation. From this point of view, their proposed countermeasure is a more efficient and general method than Mamiya's [3] countermeasure. This method could be extended to ECC public key cryptosystems which avoid the subtraction computation.

In 2006, Yi Wang [6] pointed out the disadvantage of Kim's improved countermeasure. The initial value of Kim's countermeasure for RSA is T [00] = 1 which will cause the possible attacks. Power consumption of modular multiplication of T [00] = 1 will have a big difference compared with other values. When the result change from '0' to '1' causes power, therefore, it is obvious to tell the difference between the d_is_i = 0 with other values. Yi Wang proposed an enhanced countermeasure in view of Kim's [4, 5] which keeps away from this sort of conceivable attack and turn around computation of particular exponentiation for RSA public key cryptosystems in the meantime. The essential thought of proposed countermeasures was to change the underlying estimations of T[00] which could release the sensitive data in Kim's approach. Their approach demonstrated the modified Kim's [5] countermeasure for RSA public key cryptosystems [6].

In 2006, Kaminaga *et al.* [7] proposed a conventional method where the exponent *d* was blinded and evaluated as the private key. The attackers can extract *d* because it could be implemented with a single execution result. To prevent this, they propose a countermeasure that further divides the exponent and randomizes the processing order.

Exponent and the input text performed in the conventional method are continued in the proposed method. The proposed countermeasure method was based on dividing the secret exponent d into two non-negative integers d_0 and d_1. In other words, the secret exponent is divided into the sum of two numbers as in $d = d_0 + d_1$. $S_0 = y^{d0} \bmod N$, $S_1 = y^{d1} \bmod N$ are set, and $S_0 * S_1 \bmod N = y^d \bmod N$ is obtained. d_0 is a random number less than d. To randomize the processing order of S_0 and S_1, the random number v having L(d) bits is used. Each bit of the value of v is read from the most significant bit (MSB) side. If 0, S_0 is calculated. If 1, S_1 is calculated.

In 2007, Santosh Ghosh *et al.* [8] have proposed a secure RSA implementation to resist timing and SPA attacks. They showed some implementation techniques of RSA which were insecure against different power analysis attacks. These implementations could leak the secret information while processing. They explained how simple square-multiply algorithm for RSA implementation was insecure against simple power analysis attacks. In this implementation, the exponent of plaintext was converted in binary values and then left to right binary exponentiation method was computed. Every single bit of exponent was represented by k. When the exponent k = 1 then square operation was followed by multiply operation, otherwise only square operation was performed for corresponding bit of exponent k. It was an insecure design of RSA because from the power traces the adversaries could easily find the '1's and '0's bit values of the processed exponent k. Finally they proposed square and multiply always algorithm to resist timing and SPA attack.Here, the square operation was followed by multiplication operation at every round i.e. for every bit of exponent k. This was a secured implementation of RSA against SPA and timing attacks as adversaries could not find out the position of 1's and 0's in power traces. Ji-fang *et al.* [9] had proposed a new masking technique to resist against DPA attacks in year 2009.In their work, first plaintext and exponent were masked and then processed left to right binary modular exponentiation model on masked plaintext and exponent. During each round, some inefficacy algorithm was run in between squaring and multiplication operation three times. At the end, the masked cipher text was unmasked to get the real cipher text.

In 2012 Yin [10] propose a binary modular exponentiation RSA countermeasure keeping in mind the end goal to safeguard against the comparative power analysis by partitioning the private key *e* into *n*irregular parts and arbitrarily picking one of the parts to do one unit operation every determination till the secluded exponentiation of all parts are completed. The efficiency and security of their algorithm can be improved even more by adopting the parallel computing architecture [10]. The bit length of e is noted as *le* so $e = (e_{le}; ; e_2; e_1)_2$. According to the binary method, the calculation of $M^e \bmod N$, includes *le* time for loops. Such for a loop has defined as a unit operation of the modular exponentiation. The basic conception of their defence module is randomizing the computational process of the exponentiation algorithm. They first divide the private key e into random parts, and then randomly choose one of the parts to do one unit operation each selection till the modular exponentiations of all parts are completed, and finally calculate the modular multiplication of all the modular exponentiations to get the final result.

THE RSA AND CRT-RSA

The RSA cryptosystem is utilized for both confidentiality and authentication [11]. It is the most generally utilized public key encryption algorithm. The premise of the security of the RSA algorithm is that it is numerically infeasible to factor adequately huge whole numbers. The RSA algorithm is accepted to be secure if its keys have a length of no less than 1024-bits.

The algorithm appears below:

Key Generation

1. Choose two very large random prime integers: *p* and *q*
2. Compute n and *z*, $n = pq$ and $z= (p-1)\,(q-1)$
3. Choose an integer e, $1 < e < z$ such that *gcd* (e, z) = 1
4. Compute d, $1 < d < z$ such that ed ≡ (mod z)

- The public key is (n, e) and the private key is (n, d)
- The values of p, q and z are private
- *e* is the public or encryption exponent
- *d* is the private or decryption exponent

Encryption: The cipher text C is computed by,

$$C = M^e \bmod n \qquad (1)$$

Decryption: The message M is computed by,

$$M = C^d \bmod n \qquad (2)$$

ALGORITHMS USED IN RSA

Computation of RSA involves many other algorithms for different reasons like testing the primarily, computing GCD, fast computation etc. The common algorithms used are discussed below.

Rabin Miller Algorithm

The Miller-Rabin primarily test or Rabin Miller primarily test is an algorithm for primarily testing which determines whether a given number is prime or not (12). Given two numbers P and Q to RSA system, this algorithm finds out they are prime or not. The steps involved in the algorithm are,

1. Find integers k, q such that, k > 0 and q is odd, so that $(n-1)=2^k q$
2. Select a random integer a, 1 < a < n1
3. If $a^q \bmod n = 1$ then return ("may be prime")
4. For j = 0 to k 1 do
5. If (a^{2jq} mod n = n-1)
6. Thenreturn(" may be prime ")
7. Return ("composite")

Euclidean Algorithm

The Euclidean Algorithm [13] is used for finding the GCD of two integers quickly. In mathematics, the Euclidean algorithm, or Euclid's algorithm, is a method for computing the greatest common divisor (GCD) of two numbers efficiently, the largest number that divides both of them perfectly.

In RSA the Euclidean algorithm is used to compute value of E because the value of E must be relatively prime to Z, i.e. GCD (E, Z) =1.

Euclidean Algorithm to compute GCD (a, b) is:

```
Euclid (a, b)
1. if (b=0) then
2. return a;
3. else return Euclid(b, a mod b);
```

Extended Euclidean Algorithm

It is an extension to Euclidean algorithm that not only computes GCD of integers a and b, but also the integers x and y such that, $ax+by=gcd(a,b)$.

In RSA the extended Euclidean algorithm [13] is used to compute the value of D from the value of E and Z because D= E1 mod Z. The extended Euclidean algorithm is shown next,

```
EXTENDED EUCLID (m, b)
1. (A1, A2, A3) = (1, 0, m); (B1, B2, B3) = (0, 1, b)
2. if B3 = 0 return A3 = gcd(m, b); no inverse
3. if B3 = 1
4. return A3 = gcd(m, b); b2 = b1 mod n
5. Q = A3 div B3
6. (T1, T2, T3) = (A1 Q B1, A2 Q B2, A3 Q B3)
7. (A1, A2, A3) = (B1, B2, B3)
8. (B1, B2, B3) = (T1, T2, T3)
9. go to step 2
```

Fast Exponentiation Algorithm

This algorithm is used to compute the modular exponentiation functions. For example if we need to compute the value of $c=a^n$ then write the exponent n in binary. Read the binary representation from left to right, starting with the second bit from the left. Start with the number a, and every time you read a 0 bit, square what you have got. Every time you read a 1 bit, square what you have got, multiply by a. In RSA this algorithm is used for encryption and decryption process i.e. for converting plain text to cipher text and vice-versa.

Left to Right Binary Exponentiation Algorithm:
Input: m, e and n.
Output: c = me mod n, e > 1

```
Initialization c = m if ek-1 = 1 else c=1
for j = k -1 down to 0 do
c = c * c mod n
if ( e[j] = = 1) then
c = c * m mod e
end for
return c
```

CRT-RSA

Chinese Remainder Theorem (CRT) is very effective method to speed up the decryption process for computation of the plain text or the original message from the cipher text when implemented in RSA cryptosystem. In RSA cryptosystem, $N = p * q$. By computing plain text $M=C^d \bmod N$, here one modular reduction is of size N. Using CRT in RSA cryptosystem, private key(d) is divided into d_p and d_q. By computing $m_1=C^{dp} \bmod p$ and $m_2=C^{dq} \bmod q$, m can be calculated by later merging m_1 and

m_2.Here two modular reduction is of size N_1=2. Hence, CRT is used in RSA to speed up the decryption process to get the plain text.

Implementation of CRT based RSA

The value of private exponent is very large as compared to public exponent. As a result, encryption of message will be much faster than decryption method. The value of *d*, the secret exponent cannot be made short. Therefore, CRT is used in RSA to boost up the decryption process.

Steps of implementation of CRT in RSA are as follows:

1. Let *p* and *q* be very be two large primes of nearly the same size.
2. Compute $N = p * q$.
3. Compute Euler's Totient Function, $z = (p-1)*(q-1)$.
4. Compute Public Key, *e* such that greatest common divisor, $gcd(z, e) = 1$.
5. Compute Private Key, *d* by applying $d=e^{-1} \bmod z$.
6. Given message M such that Cipher text, $C = M^e \bmod N$.
7. Compute $d_p = d \bmod (p-1)$ and $d_q = d \bmod (q-1)$ respectively.
8. Compute $p_{inv} = p^{-1} \bmod q$ and $q_{inv} = q^{-1} \bmod p$ respectively.
9. Compute signature, $m_1 = C^{dp} \bmod p$and signature, $m_2 = C^{dq} \bmod q$ respectively.
10. Compute M by the following method:

(Garner's optimization method)

$M =((q_{inv}(m_1-m_2)modp)*q) + m_2$ if m1 > m2 or $M=((q_{inv}(m_1-m_2+p)modp)*q) + m_2$ if m1 <= m2

(Gauss combination method)

$M=((m_1*q*q_{inv})+(m_2*p*p_{inv}))modN.$

In simple RSA, if the decryption process takes N times, than in CRT-RSA it will take N/2 times to compute the plain text because $m_1=c^{dp} \bmod p$ and $m_2=c^{dq} \bmod q$ runs parallel at the same time and we divide the private key (d) into two equal parts (d_p and d_q). As a result, plain text is computed very fast in CRT based RSA as compared to simple RSA.

THE MIST ALGORITHM

MIST is an effective randomized exponentiation algorithm for opposing power analysis attacks. The MIST algorithm produces haphazardly extraordinary expansion chains for performing out a specific exponentiation. The formation of long expansion chains makes it outlandish for an assailant to attack. It was first presented by Colin. D. Walter [14].

MIST involves the use of a random divisor which is not known to anyone including the attacker. This random divisor makes it more randomised and efficient thus preventing it from side channel attack, to be precise Power Analysis Attack. The choice of divisor set and addition chains for each residue R are made with security and efficiency. In case of RSA the main cost of iteration is only in the computation of Start M^d. The set of divisors used in MIST are 2, 3, 5. 4 is not used here because it produces very long addition chains in the calculation of Start M^D and M^E. To achieve a faster exponentiation very long addition chains are excluded but they might improve security.

MIST Algorithm

```
{Before proper re-scheduling of addition chain choices}
{Pre-condition: E >0}
RemE: =E;
StartM: =M;
ResultM: =1;
While RemE > 0 do
Begin
Choose a random "divisor" D;
R: = RemE mod D;
If R ≠ 0 then
ResultM: = StartM^R x ResultM;
StartM: = StartM^D;
RemE: = RemE div D;
{Loop invariant: ME = StartM^R x ResultM}
End;

{Post-condition: ResultM = M^E}
```

PROPOSED WORK

Simple as well as differential power analysis attacks have been among the most efficient and devastating attacks on some RSA-based devices. Many resisting techniques have been proposed that could prevent these attacks by securing the exponentiation algorithm which is generally targeted. Many countermeasures have already been proposed against these kinds of attacks, still it is found that these counter measures further have got some loop holes which attracts the attackers to perform cryptanalytic attacks.

In RSA system during encryption process, while computing the cipher text, i.e. $C=M^e \bmod N$ is vulnerable to power analysis attacks. But an exponentiation is basically a sequence of multiplications and squaring, but this sequence may reveal exponent bits to an attacker on an unprotected implementation. In the case of an exponentiation, the original simple power analysis is based on the fact that, if the squaring operation has a different pattern than a multiplication, the secret exponent can be directly read on the curve. For instance, in algorithm Left-to-Right Binary Exponentiation, a 0 exponent bit implies a squaring to be followed by another squaring, while a 1 bit causes a multiplication to follow a squaring [15].

So, we have proposed a modified RSA system using MIST algorithm, which is a randomized exponentiation algorithm and CRT based RSA with MIST.

MODIFIED RSA WITH MIST

The MIST algorithm generates irregular addition chains for performing a specific exponentiation. This means that power attacks which require averaging over a large number of exponentiation power traces becomes impossible. Moreover, attacks which are based on perceiving repeated use of the same pre-computed multipliers during an individual exponentiation are also additionally infeasible.

The algorithm is especially appropriate to cryptographic operations which rely upon exponentiation and which are actualized in installed frameworks, for example, smartcards. It is more proficient than the ordinary square-and-multiply operations [14].

When we apply MIST during the computation of modular exponentiation in decryption process of RSA, the power traces generated may be in randomized state in compare to RSA using left to right exponentiation algorithm. Hence attacker may not be able to guess the secret information. Figure 1 below shows the modified RSA algorithm with MIST incorporated with it.

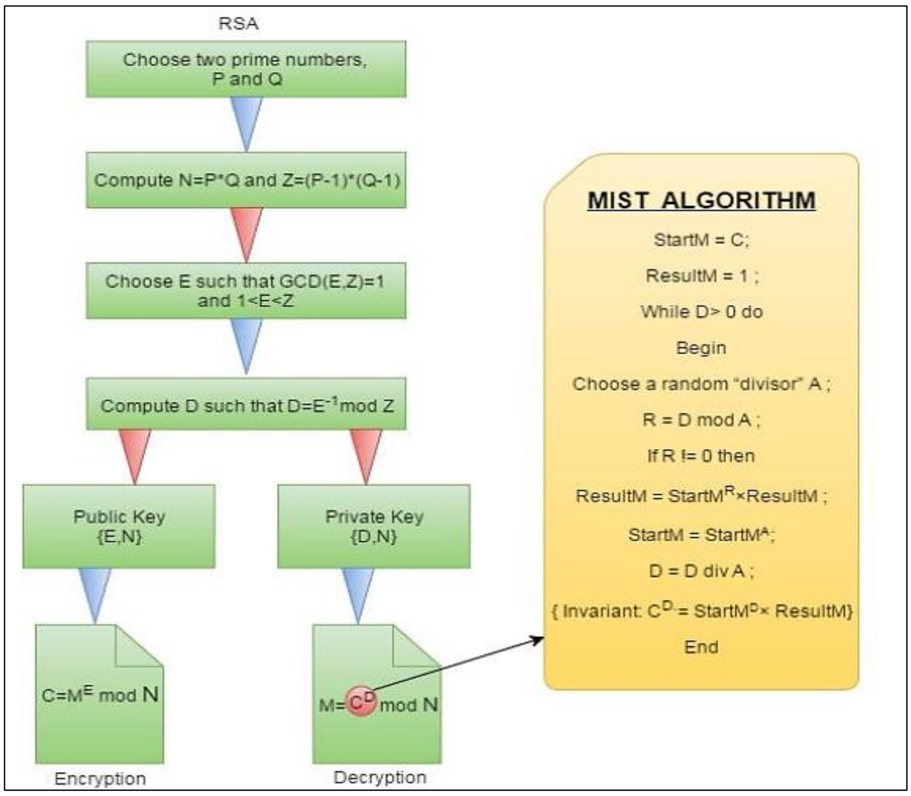

RSA
Choose two prime numbers, P and Q
Compute N=P*Q and Z=(P-1)*(Q-1)
Choose E such that GCD(E,Z)=1 and 1<E<Z
Compute D such that D=E^-1 mod Z
Public Key {E,N}
Private Key {D,N}
C=M^E mod N
Encryption
M=C^D mod N
Decryption
MIST ALGORITHM
StartM = C;
ResultM = 1 ;
While D> 0 do
Begin
Choose a random "divisor" A ;
R = D mod A ;
If R != 0 then
ResultM = StartM^R×ResultM ;
StartM = StartM^A;
D = D div A ;
{ Invariant: C^D = StartM^D× ResultM}
End

Fig. 1: *Modified RSA with MIST.*

MODIFIED CRT-RSA WITH MIST

In RSA-CRT, it is a common practice to implement the Chinese Remainder Theorem during decryption of cipher text. It results in a much faster decryption than simple modular exponentiation [16]. But in CRT based RSA also there remains a loop hole in computation of message signatures m_1 and m_2 (i.e. $m_1=c^{dp} \bmod p$ and $m_2=c^{dq} \bmod q$). These two exponentiation operations may be vulnerable to power analysis attacks. So, we may use MIST a randomized exponentiation algorithm to secure the system. Figure 2 below shows the modified CRT-RSA with MIST incorporated in it.

EXPERIMENTAL RESULTS

We have implemented Simple RSA, RSA with CRT, RSA with MIST and RSA with CRT using MIST in two different platforms Mupad (Matlab Tool) and VHDL. The motive of implementing these algorithms in Mupad are to analyze the execution time for different messages so that we can compare the performance of these algorithms with each other based on their execution time.

Implementation in Mupad

In Mupad (Matlab Tool) implementation we have computed the execution time of different message sets using various 8 bit and 16 bit keys in a workstation of the following specification:

***Table 1:** Working Environment.*

Model	HP Z230 Tower Workstation
RAM	8.00 GB
Processor	Inter(R) Xeon(R) CPU-E3-1225 v @3.20GHz
System Type	64-BIT Operating System

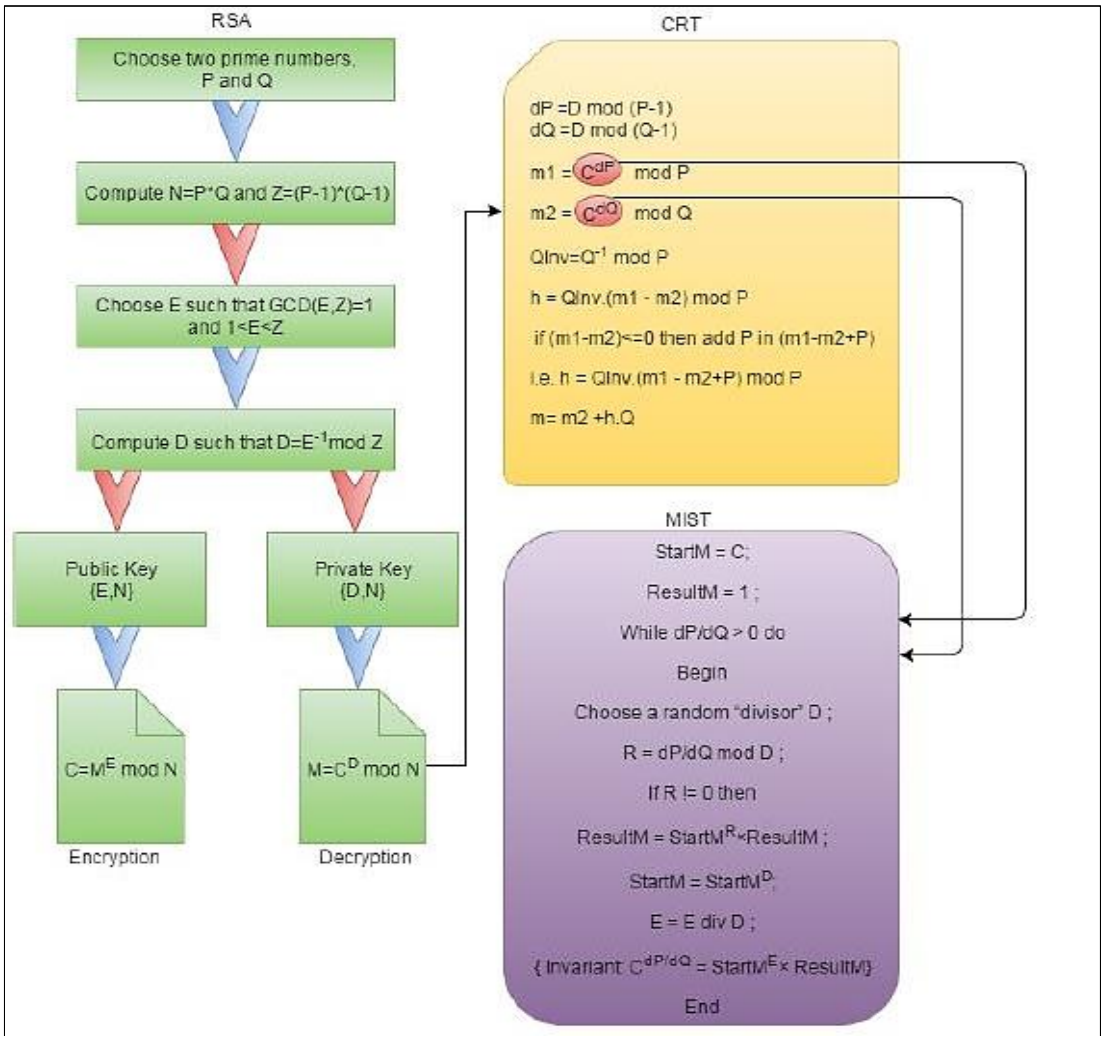


***Fig. 2:** Modified CRT-RSA with MIST.*

Two test benches have been used for testing the correctness and performance of our work. We have used PKCS standard benchmark to check the correctness of our work. We have tested with key bits from 1024 to 1030 bits with different plain texts. However, for analysing the performance of the work we have designed our own test bench of 8 bits and 16 bits (Table 1). The same benchmark was also used for VHDL. This benchmark was created by testing them individually for the RSA encryption technique implementing all the algorithms. Only after detailed analysis, they were later used in our proposed work. Tables 2 and 3 give details about our test bench. In Table 2, test bench for 8 bits key has been given and in Table 3, test bench for 16 bits key has been given. Each table has 10 different sets of *p, q, n, z, e* and *d.* For each such set we tested for 5 different plain texts in encryption and the result in decryption.

The 8 bits key test bench were used for both RSA with MIST and CRT-RSA with MIST. However, 16 bits test bench was used only for CRT-RSA with MIST, as RSA MIST consumed a lot of time.

During our simulation we have also computed the average computation time of our proposed methods. This time was the mean of the computation times for each plain text in our test bench. For 8 bit key size we have compared conventional RSA and CRT-RSA with our both proposed RSA with MIST and CRT-RSA with MIST. The results are shown in Table 4 which depicts that RSA with MIST had a higher computation time then all the other three approaches however CRT-RSA with MIST's computation time laid between the simple RSA and CRT-RSA.

For 16 bit key the average computation time of RSA with MIST was extremely high so has not been included in the results. However, CRT-RSA with MIST showed a larger computation time than the simple RSA and CRT-RSA. The results have been shown in Table 5. The computation time cost was due to the exponentiation calculation in the MIST.

Table 2: *Test Sets for 8 Bit Implementation.*

P	*Q*	*N*	*Z*	*E*	*D*
241	199	47959	47520	29	13109
163	197	32111	31752	103	3391
229	167	38243	37848	61	22957
131	157	20567	20280	251	5171
233	251	58483	58000	323	5387
131	137	17947	17680	103	14242
139	149	20711	20424	103	19171
151	157	23707	23400	101	130901
163	167	27221	26892	103	20887
173	179	30967	30616	109	11797

Table 3: *Test Sets for 16 Bit Implementation.*

P	***Q***	***N***	***Z***	***E***	***D***
32999	33181	1094939819	1094873640	493	1010481757
32993	33203	1089527839	1089461024	243	546972603
33161	33203	1101044683	11009783320	539	526998899
33151	33199	1100580049	1100583700	1019	964434497
33053	33199	1095673897	1095607696	999	545062087
32771	32779	1074200609	107435060	119	857502779
32783	32789	1074921787	1074853216	721	994908233
32797	32801	10757743397	10757008800	283	22806547
32803	32831	1076955293	1076889660	263	217015787
32833	32839	10782028887	1078137216	233	828268505

Table 4: *Computation Time for 8 Bit Implementation.*

Simple RSA	**RSA-CRT**	**RSA-MIST**	**CRT-RSA-MIST**
0.21216136	0.21216136	0.29320474	0.18632126
0.1716011	0.16224104	0.2028013	0.173201
0.2148023	0.17336156	0.25056226	0.22328688
0.22208258	0.20428208	0.23744324	0.21696316
0.25694298	0.17196266	0.286049108	0.245464394
0.38336306	0.29240373	0.520005	0.4612047
0.27929238	0.23712352	0.47328638	0.33304534
0.40664494	0.23592482	0.48744774	0.33576596
0.26848558	0.18464544	0.33552892	0.23649858
0.28024614	0.20304584	0.39064994	0.246073

Figure 3 shows the plot of average computation time versus the sets of test bench for 8 bit key. Similarly, plot 4 shows the average computation time versus the sets of test bench for 16 bit key (Figure 4). These graphs can clearly show the average computation time for simple RSA, CRT- RSA and our both proposed methods.

IMPLEMENTATION IN VHDL

VHDL stands for VHSIC (Very High Speed Integrated Circuits) Hardware Description Language. An advanced framework in VHDL comprises of an outline element that can contain different elements that are then considered parts of the best level substance. Every substance is displayed by an element revelation and an engineering body.

We have used Xilinx ISE 9.2 version for the implementation. The main motive of implementation in VHDL was to get the power consumptions details during various processes in different algorithms, so that we can compare the randomness of our proposed algorithm with Simple RSA and RSA-CRT. However, due to lack of adequate equipments we could not obtain our desired results. So, we only present a model of the VHDL modules. Figures 5 and 6 show the VHDL module for RSA and CRT- RSA.

Table 5: *Computation Time for 16 Bit Implementation.*

Simple RSA	RSA-CRT	CRT-RSA-MIST
0.13728088	0.1248008	0.2964019
0.13104084	0.11856076	0.45864294
0.12168078	0.11232072	0.31824186
0.15152092	0.11856076	0.27768178
0.14694572	0.12792082	0.20904134
0.12168078	0.11232072	0.37128238
0.11232072	0.10608068	0.26520166
0.12344074	0.1148008	0.24336156
0.1152007	0.09816086	0.28376246
0.12580078	0.1128008	0.26872262

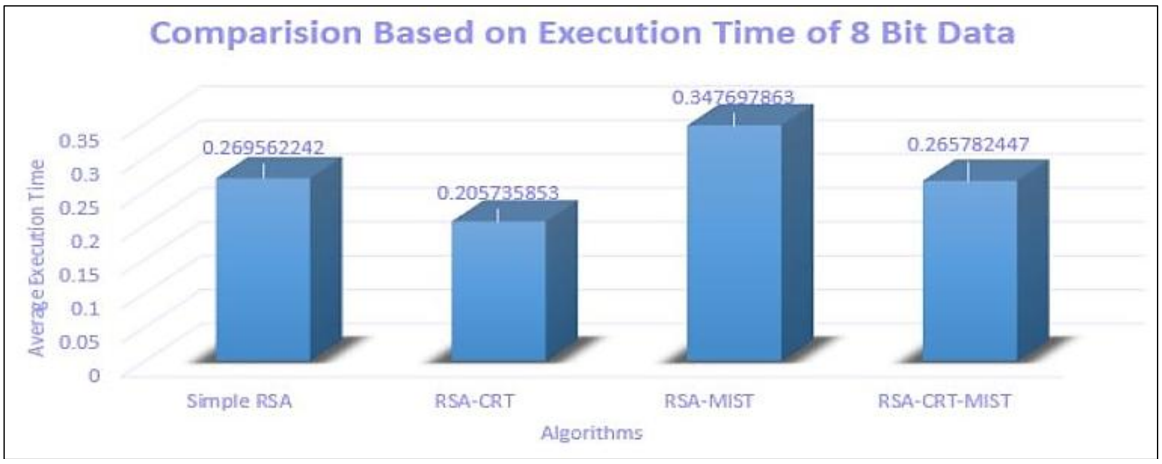


Fig. 3: *Average Computation Time for 8 Bit Key.*

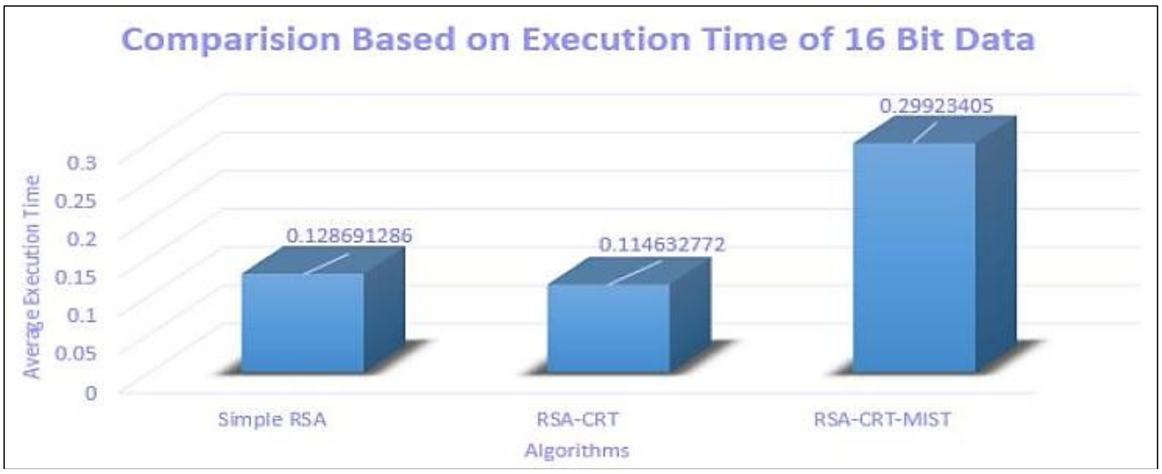


Fig. 4: *Average Computation Time For 16 Bit Key.*

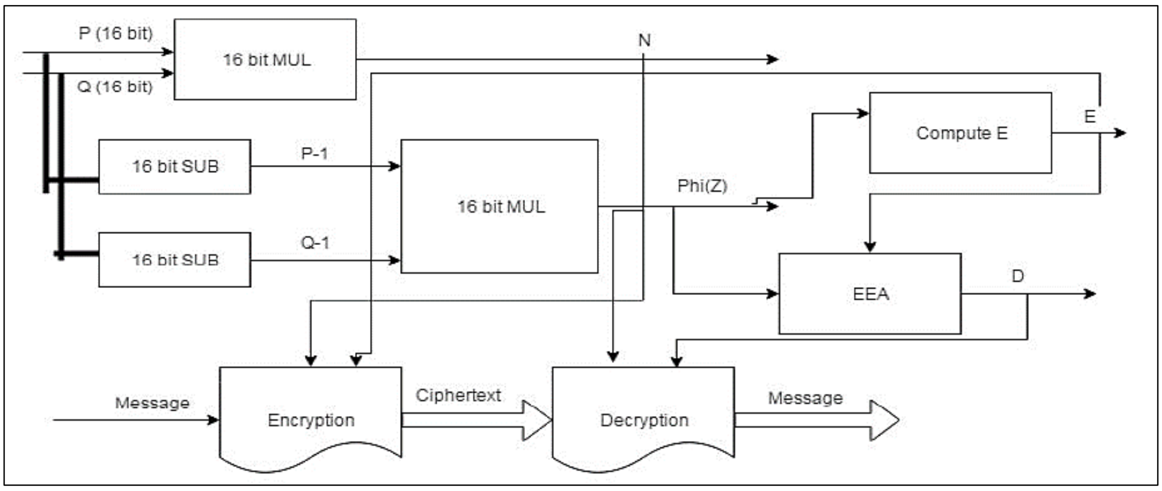
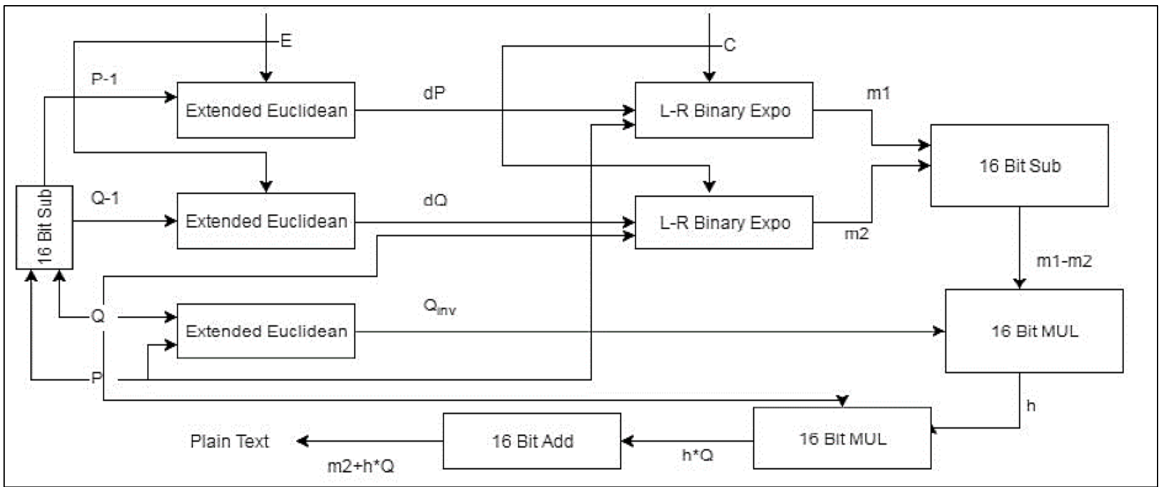
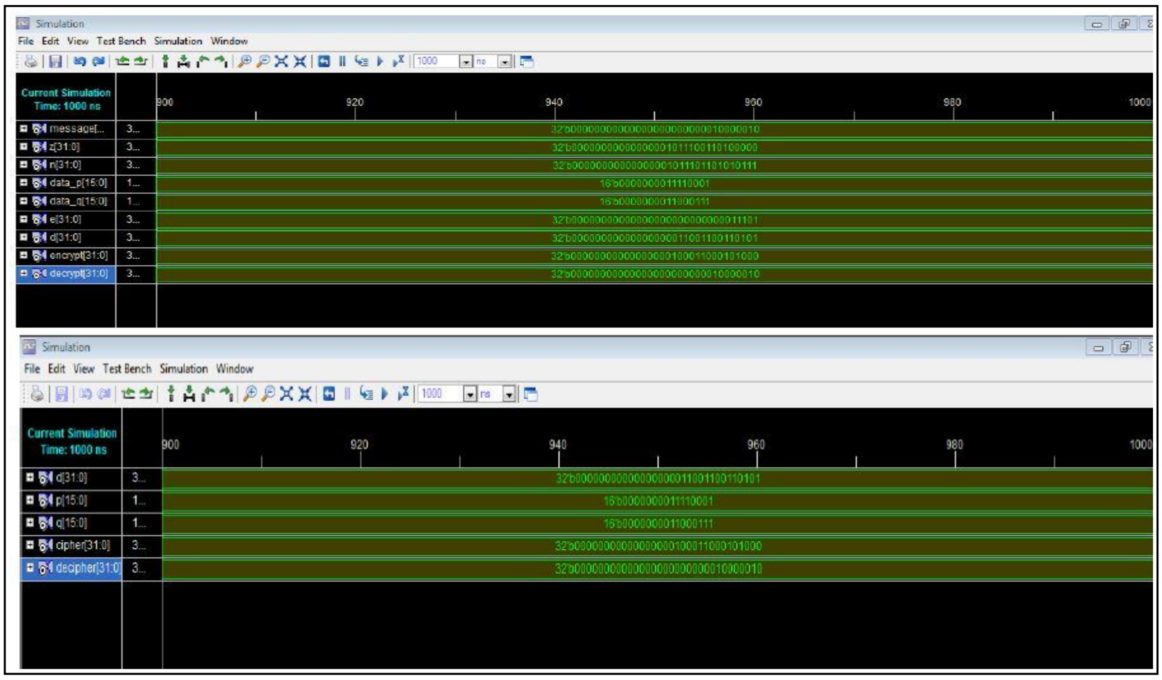


Fig. 5: *VHDL Module For RSA.*




Simulation
File Edit View Test Bench Simulation Window
1000
ns
Current Simulation Time: 1000 ns
900
920
940
960
980
1000

```
message[...  3...  32'b00000000000000000000000010000010
z[31:0]      3...  32'b00000000000000001011100110100000
n[31:0]      3...  32'b00000000000000001011101101010111
data_p[15:0] 1...  16'b0000000011110001
data_q[15:0] 1...  16'b0000000011000111
e[31:0]      3...  32'b00000000000000000000000000011101
d[31:0]      3...  32'b00000000000000000011001100110101
encrypt[31:0] 3... 32'b00000000000000000100011000101000
decrypt[31:0] 3... 32'b00000000000000000000000010000010
```

```
d[31:0]      3...  32'b00000000000000000011001100110101
p[15:0]      1...  16'b0000000011110001
q[15:0]      1...  16'b0000000011000111
cipher[31:0] 3...  32'b00000000000000000100011000101000
decipher[31:0] 3... 32'b00000000000000000000000010000010
```

Fig. 7: *VHDL Output for RSA And CRT-RSA.*

CONCLUSION

In this paper we have implemented two modified RSA implementations, RSA with MIST and CRT-RSA with MIST. We aimed to associate MIST in RSA implementation in order to provide randomization so that it can resist power analysis attacks (Figure 7). From our experimental results we can conclude that RSA with MIST provide additional security but with a huge time cost. But, CRT-RSA with MIST can be very useful both from performance as well as security perspective. We have also designed the VHDL modules for our proposed approaches. But calculating the actual power consumption details remains as the future work for our proposed methods.

REFERENCES

1. Mangard S. *et al. Power analysis attacks: Revealing the secrets of smart cards,* Springer Science Media House. 2008; 31.
2. Mahanta HJ *et al. Information System Design and Intelligent Application,* Springer. 2015; 2: 349–358 p.
3. Mamiya H. *et al. CHES 2004.* LNCS 3156. Springer-Verlag, 342–256p.
4. Kim CK *et al. International Conference on Computational Science and its Application,* Springer Berlin Heidelberg, 2004; 150–158p.
5. Kim CK. *et al. IACR Cryptology ePrint Archive* 2005; 22.
6. Wang Y *et al. IEEE Asia Pacific Conference on Circuits and Systems,* 2006.
7. Kaminaga M *et al. Electronics and Communications in Japan (Part III: Fundamental Electronic Science),* 2006; 89(8): 10–20p
8. Ghosh S. *IEEE Region 10 Conference TENCON, 2007*
9. Jin J *et al. Fifth International Conference on Information Assurance and Security,* 2009; 2.
10. Yin X *et al. IEEE International Conference on Intelligent Control, Automatic Detection and High-End Equipment,* 2012.
11. Rivest RL *et al. Communications of the ACM,* 1978; 21(2): 120–126p
12. Arnault F. *Mathematics of Computation,* 1995; 64(209): 355–361p.
13. Stallings W. *Cryptography and Network Security,* 2006. Pearson Education India
14. Colin D. *Topics in Cryptology CT-RSA,* 2002. Springer Berlin Heidelberg. 53–66p.
15. Ha J. *et al. Journal of Internet Services and Information Security (JISIS),* 2014; 4(4): 38–51p.
16. Shinde G. N. *et al. International Conference on Computational Experimental Engineering and Sciences,* 2008; 5(4).

Cite this Article
Hridoy Jyoti Mahanta, Ajoy Kumar Khan. Implementation of RSA and CRT-RSA with MIST to Resist Power Analysis Attacks. *Journal of VLSI Design Tools & Technology*. 2017; 7(3): 1–12p.