Security through Elliptic Curves for Wireless Network in Mobile Devices

E.KESAVULU REDDY P.GOVINDA RAJULU

Abstract: **The basic principle is "A function is easy to evaluate but its invert is infeasible unless a secret key is known".**

It is mathematically proved that security of cryptographic does not imply its implementation security of system against Side-channel Attacks. The security of system lies in the difficulty of extracting k from P and Q. It is essential to secure the implementation of cryptosystems in embedded devices against side-channel attacks. These attacks monitor the power consumption or the Electromagnetic emanations of a device Ex smart cards or mobile devices. The attacker's goal is to retrieve partial or full information about a long-term key that is employed in several ECSM executions.

We are implementing a secret key avoid to retrieving the valuable information by the attacker through Simple Power Analysis Attacks.

Key words: **Elliptic curve cryptography, Simple power Analysis, Differential Power analysis**

I. Introduction

Elliptic curve cryptosystems (ECCs) are suitable for implementation on devices with limited memory and computational capability such as smart cards and also with limited power such as wireless handheld devices. This is due to the fact that elliptic curves over large finite fields provide the same security level as other cryptosystems such as RSA for much smaller key sizes.

Considering power analysis attacks, there are two main types that were presented by Kocher et al. These are simple and differential power analysis attacks (referred to as SPA and DPA respectively).

Both of them are based on monitoring the power consumption of a cryptographic token while executing an algorithm that manipulates the secret key.

The traces of the measured power are then analyzed to obtain significant information about the key. In some cases the key can be totally compromised and in others the search space of the key can be reduced to a computationally affordable size. In SPA, a single power trace can reveal large features of the algorithm being executed such as the iterations of the loop. Moreover, cryptosystem-specific operations such as point doubling and adding in ECCs can be identified [3]. In order to resist this SPA attack, the steps of the algorithm need to be uniform across different executions.

Asst Prof. E.KESAVULU REDDY, Assistant Professor, Dept.of Computer Science, S.V.University College of CM & CS, Tirupati, A P-India-517502, Mobile:+91 9866430097
(e-mail: ekreddysvu2008@gmail.com)
Prof. P.GOVINDA RAJULU, Professor, Dept.of Computer Science, S.V.university College of CM & CS,Tirupati, AP-India-517502,Phone:0877-2249916,Mobile:+919912349770
(e-mail: pgovindarajulu@yahoo.com)

Hence, DPA attacks are, in general, more powerful than the SPA attack. Randomization of the data processed at some instant is essential in resisting this type of attacks.

Electromagnetic emanations present another powerful side channel since the information is leaked from the device via more than one channel and is a function of space as well as of time. In [2], Agrawal et al. presented both simple and differential electromagnetic analysis attacks on smart cards and on a Palm pilot . In these attacks they conclude that software countermeasures rely on signal information reduction, which is achieved by "randomization and/or frequent key refreshing within the computation", which agrees with the concept of resisting DPA attacks.

The point addition operation consists of finite field operations carried in the underlying field K. We denote the field inversion by I, the multiplication by M, the squaring by S. The point addition is denoted by A. When the two operands of the addition are the same point, the operation is referred to as point doubling and is denoted by D.

II. Window Methods

This method is sometimes referred to as m−ary method. There are different versions of window methods . What is common among them is that, if the window width is w, some multiples of the point P up to $(2^w - 1)P$ are precomputed and stored and k is processed w bits at a time. k is recoded to the radix 2^w. k can be recoded in a way so that the average density of the nonzero digits in the recoding is $1/(w + \xi)$, where $0 \le \xi \le 2$ depends on the algorithm. Let the number of precomputed points be t, in the Precomputation stage, each point requires either a doubling or an addition to be computed also depending on the algorithm.

This ECSM method is suitable for unknown or fixed point P. The cost is Storage: t points, where $2^{w-2} \le t \le 2^{w-1}$ depending on the algorithm.

Precomputation: t point operations (A or D).

Expected running time:

$$(n-1)\,D + n\frac{n}{w+\xi}\,A,$$

where $0 \le \xi \le 2$ depending on the algorithm. Note that the number of doubling is between n − w and n − 1.

A. Simultaneous multiple point multiplication

This method is used to compute kP +lS where P may be a known point. This algorithm was referred to as Shamir's trick in [10]. If k and l are n-bit integers, then their binary representations are written in a 2 × n matrix called the exponent array. Given width w, the values iP +jS are calculated for $0 \le i, j < 2^w$. Now the algorithm performs d = $\lceil n/w \rceil$ iterations. In every iteration, the accumulator point is

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

doubled w times and the current 2 × w window over the exponent array determines the precomputed point that is to be added to the accumulator.

Algorithm *A. Simultaneous multiple point multiplication (Shamir-Strauss method)*

Input: Window width w, d = $\lceil n/w \rceil$,

k = $(K_{d-1}, \ldots, K_1, K_0)_{2^w}$ l= $(L_{d-1}, .., L_1, L_0)_{2^w}$, and P, S ∈ E($F_q$). 3Also according to [Ber01], it is originally due to Straus [Str64].

Output: kP + lS.

1. Precomputation. Compute iP + jS for all i, j ∈ [0, $2^w - 1$].

2. Q ← K_{d-1} P + L_{d-1} S.

3. for i from d − 2 down to 0 do

3.1 Q ← 2^w Q.

3.2 Q ← Q + (K_i P + L_i S).

4. Return(Q).

Storage: $2^{2w} - 1$ points. For w = 1, 3 points.

For w = 2, 15 points.

Precomputation:

$(2^{2(w-1)} - 2^{w-1})$ D + $(3 \cdot 2^{2(w-1)} - 2^{w-1} - 1)$ A.

For w = 1, 1 A.

For w = 2, 1 D + 11 A.

Expected running time:

$(d-1)w\,D + \frac{(2^{2w}-1}{2^{2w}} d - 1)$ A.

For w = 1, (n − 1) D +($\frac{3}{4}$ n − 1) A.

For w = 2, (n − 1) D +($\frac{15}{32}$ n -1) A.

Using sliding windows can save about 1 4 of the precomputed points and decrease the number of additions to $\frac{n}{w+(1/3)}$, which is about 9% saving for w ∈ {2, 3}.

B. Interleaving method

This method is also a multiple point multiplication method, and we want to compute $\sum k^j P_j$ for points P_j and integers K^j. In the comb and simultaneous multiplication methods, each of the precomputed values is a sum of the multiples of the input points. In the interleaving method, each precomputed value is simply a multiple of one of the input points. Hence, the required storage and the number of point additions at the precomputation phase is decreased at the expense of the number of point additions in the main loop. This method is flexible in that each k_j can have a different representation, e.g., different window size, as if a separate execution of a window method is performed for each k^j P_j with the doubling step performed jointly on a common accumulator, as shown in [12]. As an illustration, we provide the following algorithm that *computes kP + lS where both k and l are represented to the same base* 2^w.

Algorithm *B. Interleaving method*

Input: width w, d = $\lceil n/w \rceil$,

k = $(K_{d-1}, \ldots, K_1, K_0)_{2^w}$ l= $(L_{d-1}, \ldots, L_1, L_0)_{2^w}$, and P, S ∈ E($F_q$).

Output: kP + lS.

1. Precomputation. Compute iP and iS for all i ∈ [0, $2^w - 1$].

2. Q ← K_{d-1} P.

3. Q ← Q L_{d-1} S.

4. for i from d − 2 down to 0 do

4.1 Q ← 2^w Q.

4.2 Q ← Q + K_i P.

4.3 Q ← Q + L_i S.

5. Return(Q).

Storage: $2^{w+1} - 2$ points.

Precomputation: 2(w-1) D + $2(2^w - w - 1)$ A.

Expected running time: w(d − 1) D + (2d − 1). $\frac{(2^{2w}-1)}{2^{2w}}$ A

In general, if different basis and/or representations are used for k and l, we have

Storage: 2t points, where $2^{w-2} \le t \le 2^{w-1}$ depending on the particular window algorithm used as discussed in Section B

Precomputation: 2t point operations (A or D).

Expected running time: (n − 1) D + 2 $\frac{n}{w+i}$ A, where $1 \le i \le 2$ depending on the algorithm

C. SPA Attack on ECCs and its Countermeasures

Coron [3] has transferred the power analysis attacks to ECCs and has shown that an unaware implementation of EC operations can easily be exploited to mount an SPA attack. Moreover, it may also enable to recognize the exact instruction that has been executed. For example, if the difference in power consumption between point doubling (D) and point addition (A) is obvious in their respective power traces, then, by investigating one power trace of a complete execution of a double-and-add algorithm, the bits of the scalar k are revealed. That is, whenever a D is followed by A, the corresponding bit is k_i = 1, otherwise if D is followed by another D, then k_i = 0. This sequence of point operations is referred to as the DA sequence.

Window methods process the key on a digit (window) level. The basic version of this method, that is where = 0 in Section A, is inherently uniform since in most iterations, w D operations are followed by 1 A, except for possibly when the digit is 0. Therefore, fixed-sequence window methods were proposed in order to recode the digits of the key such that the digit set does not include 0.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

D. DPA Attack on ECCs and its Countermeasures

As for the SPA attack, Kocher et al. were first to introduce the DPA attack on a smart card implementation of DES. Techniques to strengthen the attack and a theoretical basis for it were presented by Messerges et al. in [3]. Coron applied the DPA attack to ECCs [3].

In order to resist DPA attacks, it is important to randomize the value of the long-term key involved in the ECSM across the different executions. Some of the countermeasures that were based on randomizing the key representation were proven to be inadequate since the intermediate point computed in the accumulator Q at certain iteration remained one of two possible values. The constancy of the value of this intermediate point is an integral part in the success of first-order DPA attacks.

A potential DPA countermeasure is known as key splitting. It is based on randomly splitting the key into two parts such that each part is different in every ECSM execution. An additive splitting using subtraction is attributed to Clavier and Joye. It is based on computing

$$kP = (k - r)P + rP, \quad (I)$$

The authors mention that the idea of splitting the data was abstracted in [5]. where r is a n-bit random integer, that is, of the same bit length as k. Alternatively, Ciet and Joye [8] suggest the following additive splitting using division, that is, k is written as

$$k = \lfloor k/r \rfloor + (k \bmod r). \quad (1)$$

Hence, if we let $k_1 = (k \bmod r)$, $k_2 2 = \lfloor k/r \rfloor$ and $S = rP$, we can compute $KP = k_1 p + k_2 P$ (2)

where the bit length of r is n/2. They also suggest that (2) should be evaluated with Shamir-Strauss method as in Algorithm C. However, they did not mention whether the same algorithm should be used to evaluate (1). The following multiplicative splitting was proposed by Trichina and Bellezza [0] where r is a random integer invertible modulo u, the order of P. The scalar multiplication kP is then evaluated as

$$kP = [kr^{-1} \pmod u](rp) \quad (3)$$

To evaluate (3), two scalar multiplications are needed; first R = rP is computed, then kr^{-1} R is computed.

III. Key Splitting Methods

A. Introduction

We discuss different the forms of key splitting along with their strengths and weaknesses. We also discuss the candidate SPA-resistant algorithms and compare the resulting performance when combined with each form of key splitting. At the end of the chapter, we present countermeasures to DPA attacks on the ECDSA and the ECMQV algorithms.

This approach was suggested by Clavier and Joye in [5] and revisited by Ciet [6] as follows. In order to compute the point kP, the n-bit key k is written as

$$k = k_1 + k_2,$$

such that $k_1 = k - r$ and $k_2 = r$, where r is a random integer of length n bits. Then kP is computed as

$$kP = k_1 P + k_2 P. \quad (1)$$

It is important to note that each of the terms of should be evaluated separately and their results combined at the end using point addition. That is the multiple-point multiplication methods that use a common *accumulator to save doubling operations.* Whether at the bit level (w ≡ 1) or window level (w > 1)-should not be used, even when a countermeasure against SPA is employed. This observation is based on the following lemma. Let $_{b\to a}$ denote $\lfloor k(mod\ 2^{b+1})2^a \rfloor$or, simply, the bits of k from bit position b down to bit position a, with $b \geq a$.

Lemma 3.2 Let splitting scheme I at the end of some iteration j, $0 < j \leq n - 1$, there are only two possible values for Q, those are $[k_{n-I-j}]$ P or $[k_{n-I-j} - 1]$ P.

Proof. Algorithm C—and similarly computes the required point by scanning

$k_1 = (k_{1\ n-1}, \ldots, k_{1\ 0})2$ and

$k_2 = (k_{2\ n-1}, \ldots, k_{2\ 0})_2$ from the most significant end down to the least significant end. Hence, at the end of iteration j, the accumulator Q contains the value

$$Q = k_{1n-I-j} P + k_{2n-I-j} P \quad (2)$$

$$= [k_{1n-I-j} + k_{2n-I-j}] P.$$

We can write k, k_1 and k_2 as

$$k = k_{n-I-j^j} 2^j + k_{j-1-0}$$

$$k_i = k_{i\ n-I-j^j} 2^j + k_{i\ j-1-0} \quad (3)$$

Since $k = k_1 + k_2$ we have

$$k_{1\ j-I-0} + k_{2\ j-I-0} j = k_{j-I-0} + b\ 2^j \text{ where } b \in \{0, 1\} \quad (4)$$

and

$$k_{1\ n-I\to j} + k_{2\ n-I\to j} = k_{n-I\to j} - b$$

The DPA attack would proceed in the same way, whether the algorithm processes a single bit or a digit per iteration, though it would be more involved in the latter case depending on the digit size. The attacker can double the number of traces gathered and compute the necessary intermediate points as if there was no countermeasure in place.

B.Modular division:

In the following algorithm, a and b are integers internally represented each by an array of w-bit digits. The length of each array is $d = \lceil n/w \rceil$ digits. Note that for the modular inversion, as mentioned by Savas and Koc , b needs not be less than the modulus u, but be in $[1, 2^m - 1]$, where m = dw. Also note that the values $R^2 \bmod u$, where $R = 2^m$, and u' are computed once per modulus, i.e., per curve.

Algorithm B. Modular division

Input: u: a n-bit prime, $d = \lceil n/w \rceil$, m = dw, $R^2 \pmod u = (2m)^2 \pmod u$, $u' = u^{-1} \bmod 2^w$, $a \in [1, p - 1]$ and $b \in [1, 2^m - 1]$.

Output: $ab^{-1} \pmod u$.

1. Compute $b^{-1} R \pmod u$.
2. Compute $x = a(b^{-1} R)R^{-1} \pmod u$
3. Return(x).

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Algorithm C Montgomery multiplication

Input: u: a n-bit prime, $d = \lceil n/w \rceil$, m = dw, u' = u^{-1} mod 2^w, x = (x_{d-1} . . . , x0)2^w and

y = (y_{d-1} . . . y0)2^w.

Output: $xy2^{-m}$ (mod u).

1. A ← 0. // A = (ad, a_{d-1}, . . . , a0)2w
2. for i from 0 to d − 1 do
2.1 ui ← $(a_0 + x_i y_0) \bmod 2^w$
2.2 A ← $(A + x_i y_0 + u_i m)/ 2^w$
3. if (A > u) then
A A ← u.
4. Return(A).

The following algorithm was presented by Savas and Ko¸c as the modified Kaliski-Montgomery Inverse.

Algorithm D. Montgomery inversion

Input: u: a n-bit prime, $d = \lceil n/w \rceil$, m = dw, R^2 (mod u) = $(2^m)^2$ (mod u), u' = u^{-1} mod 2^w and b ∈ [1, 2^m − 1].

Output: b^{-1} R (mod u).

1. Compute f and x = $b^{-1}2^f$ Where n ≤ f ≤ m + n.
2. if (f ≤ m) then
2.1 x ← xR^2R^{-1} (mod u)
x = $b^{-1}2^{m+f}$ (mod u)
2.2 f f + m. // f > m, x = b−12f (mod u)
3. x ← $x2^{2m-f}R^{-1}$ (mod u) using Algorithm 6.5.
// x = $b^{-1}2^f 2^{2m-f} 2^{-m} = b^{-1}2^m$ (mod u)
4. Return(x).

IV. Existing System

Nevine Maurice Ebied's modified Almost Montgomery inverse algorithm to be resistant to SPA attacks. In the following algorithm. Swap Address(c, d) denotes interchanging the memory addresses of the integer's c and d. This is an inexpensive operation, hence its usage as a dummy operation to balance the branches of the main loop. We implemented the "if" statement in steps 3.4 and 3.5 such that the number of conditions checked per loop iteration is always three. In assembly language, this can be easily ensured. Written in Java, step 3.4 is implemented as

if((xLSb = = 0) && (xLSb = = 0) && (xLSb = = 0)).

If the condition is false, due to short-circuit evaluation, the flow control will move to the following "if" after the first check, otherwise, it will perform the check three times. The following "if"—step 3.5—is similar but with the condition checked only two times

if((yLSb = = 0) && (yLSb = = 0)).

Algorithm 3.7. Almost Montgomery inverse :Input: u: a n-bit prime,

$d = \lceil n/w \rceil$, m = dw and b ∈ [1, 2^m − 1].

Output: f and $b^{-1}2^f$ (mod u), where n ≤ f ≤ m + n.

1. x ← u; y ← b; r ← 0; s ← 1.
2. f ← 0.
3. while (v > 0) do
3.1 U ← x − y; V ← −U.
3.2 T ← r + s.
3.3 f ← f + 1.
3.4 if (((lsb(x) = 0))) then // This "if" is special
SwapAddress(x, U); SwapAddress
(x, U) // dummy
SHR(x); SHL(s).
3.5 else if ((lsb(y) = 0)) then // This "if" is special
SwapAddress(y, V); SwapAddress
(y, V) // dummy
SHR(y); SHL(r).
3.6 else if (V >= 0) then
SwapAddress(y, V);
SwapAddress(s, T)
SHR(y); SHL(r).
3.7 else
SwapAddress(x, U);
SwapAddress(r, T)
SHR(x); SHL(s).
4. T ← u − r; V ← u + T .
5. if (T > 0) then
Return(f, T)
else
Return (f, V).

The drawback of this algorithm is that an SPA of the number of iterations of the main loop directly leaks the value of f. If f is uniformly distributed, the search space of b s reduced from 2^w to $2^{m-\log 2^m}$, which is not a significant reduction. It is interesting to study how f is actually distributed.

A. Proposed system

We modified the Nevine Maurice Ebied's Almost Montgomery inverse and A SECRET KEY of [Savas and Koc] to be resistant to SPA attacks as in the following algorithm.

Algorithm A. EKR Modified Montgomery Inversion

Input: u: a n-bit prime, $d = \lceil n/w \rceil$,

m = dw, R^2 (mod u) = $(2^m)^2$ (mod u), u' = u^{-1} mod 2^w and b ∈ [1, 2^m − 1], t is Secret key.

t: No of precomputed points 1≤ t ≤n

W: Window width least significant of bit

2^{w-z} $t \le 2^w - 1$

Output: b^{-1} R (mod u).

1. Select a number b such that $(b, 2^m) = 1$
2. Compute b^{-1} such that $bb^{-1} \equiv 1 (\bmod 2^m)$
3. If f > m then x = $b^{-1}2^f$ (mod u)
∵ x = $b^{-1}2^f$ (mod u)
4. If f ≤ m then
5. x ← $R^2R^{-1} (\bmod u)$ ∵ R = 2^m
6. x=$b^{-1}2^{m+f}$ (mod u) f ← m+f
7. Return(x)

In EKR modified Montgomery Inverse Algorithm of Savas and Koc , we select f such that gcd $(b, 2^m) = 1$, m ≤ f ≤ m +n . So b is not reduced from 2^m to $2^{m-\log 2^m}$. Therefore this is significant reduction and hence f is not uniformly distributed and it can't leaks the value

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

B. Source Code:

```
package javaapplication1;
import java.util.ArrayList;
import java.util.Scanner;
public class Main {
public static void main(String[] args)
   {
Scanner s=new Scanner(System.in);
System.out.println("Enter a Prime Integer(U):");
int u=s.nextInt();
String binU=Integer.toBinaryString(u);
int n=binU.length();
System.out.println("Enter a secrete Key(t):");
int t=s.nextInt();
double w=0;
for(int wi=1;wi<=t;wi++)
{
   int i1=(int)Math.pow(2,wi-2);
   int i2=(int)Math.pow(2,wi)-1;
   if(i1<=t && t<=i2)
   {
      w=wi;
      break;
   }
}

System.out.println("Window width (W): "+w);
int d=(int)Math.ceil(n/w);
System.out.println("Length of Each Array (d) :"+d);
int m=(int)(d*w);
System.out.println("....(m) :"+m);
int on1=(int)Math.floor(Math.pow(2,m-1));
int n2=(int)Math.pow(2,m);
for(int i=m;i<on1;i++)
{int n1=i;
while(n1!=n2)
 {
        if(n1>n2)
                n1=n1-n2;
        else
                n2=n2-n1;
 }

 //System.out.println("GCD of two number is "+n1+"and i
is "+i);
if(n1==1){on1=i;break;}
}
System.out.println("B="+on1);
int b=on1;

int b_inverse=0;
for(int i=1;i<b;i++)
{
int t2m=(int)Math.pow(2,m);
int temp=t2m*i;

temp=temp+1;
b_inverse=temp/b;
if((temp%b)==0)
{
break;
}
}
System.out.println("B inverse: "+b_inverse);

ArrayList<Integer> af=new ArrayList<Integer>();
for(int i=n;i<=(m+n);i++)
{   af.add(i);
}
int x=0;
for(int f=n;f<=(n+m);f++)
{
if(f>m)
{
 x=(int)(b_inverse*Math.pow(2, f))/u;
System.out.println("f="+f+"\nx="+x);
}
else if (f<m)
{
   x=(int)(b_inverse*(Math.pow(2,m+f)))/u;

System.out.println("f="+af.get(f)+"\nx="+x);
}

}

   }

}
```

C. Input and output

```
Enter a Prime Integer(U):
111
Enter a secrete Key(t):
117
Window width (W): 7.0
Length of Each Array (d) :1
....(m) :7
B=7
B inverse: 55
f=8
x=126
f=9
x=253
f=10
x=507
f=11
x=1014
f=12
x=2029
f=13
x=4059
f=14
x=8118
BUILD SUCCESSFUL (total time: 29 seconds)
```

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

V. Conclusion

We modified the Nevine Maurice Ebied's Almost Montgomery inverse and A New variant of [ScKK00] of Montgomery Inversion i.e is EKR Modified Montgomery Algorithm to be resistant to SPA attacks . In EKR Modified Montgomery Inverse Algorithm to eliminate the number of Iterations of the main loop directly leaks the value of f and also it is mathematically proved that f is uniformly distributed with a significant reduction

A function that is easy to evaluate but its inverse is infeasible unless the secret key t is known. So the attacker can not guess the key (t) to retrieve the valuable information in smart cards and mobile devices.

REFERENCES

[1]. Nevine Maurice Ebied's Key Randomization Counter Measures To Power Analysis Attacks On Elliptic Curve Cryptosystems Ph.D. thesis, University of Waterloo, Ontario, Canada, 2007

[2]. D. Agrawal, B. Archambeault, J. R. Rao & P. Rohatgi. The EM Side-Channel(s): Attacks and Assessment Methodologies. Internet Security Group, IBM Watson Research Center.ps. 2, 3

[3]. J.-S. Coron. "Resistance against differential power analysis for elliptic curve cryptosystems". In Cryptographic Hardware and Embedded Systems –CHES '99, LNCS, vol. 1717, pp. 292–302. Springer-Verlag, 1999. 2, 22, 24, 158, 170, 180, 181, 186

[4]. S. Chari, C. S. Jutla, J. R. Rao & P. Rohatgi. "Towards sound approaches to counteract power-analysis attacks." In Advances in Cryptology – CRYPTO '99, LNCS, vol. 1666, pp. 398–412. Springer-Verlag, 1999. 24

[5]. C. Clavier & M. Joye. "Universal exponentiation algorithm a first step towards provable SPA-resistance". In Cryptographic Hardware and Embedded Systems – CHES '01, LNCS, vol. 2162, pp. 300–308. Springer-Verlag, 2001. 4, 24, 120

[6]. M. Ciet, J.-J. Quisquater & F. Sica. "Preventing differential analysis in GLV elliptic curve scalar multiplication". In Cryptographic Hardware and Embedded Systems – CHES '02, LNCS, vol. 2523, pp. 540–550. Springer- Verlag, 2003. 4, 25, 126, 173, 179

[7]. T. ElGamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". IEEE Transactions on Information Theory, 31(4):469– 472, 1985.

[8]. J. Ha & S. Moon. "Randomized signed-scalar multiplication of ECC to resist power attacks". In Cryptographic Hardware and Embedded Systems – CHES '02, LNCS, vol. 2523, pp. 551–563. Springer-Verlag, 2002. 3, 24, 27, 31, 40, 58, 70, 75, 93, 101, 173

[9]. T. S. Messerges, E. A. Dabbish & R. H. Sloan. "Investigations of power analysis attacks on smart cards". In USENIX Workshop on Smart- card Technology, pp. 151–161. May 1999. 2, 24, 172

[10]. B. M¨oller. "Securing elliptic curve point multiplication against sidechannel attacks". In International Security Conference – ISC '01, LNCS, vol. 2200, pp. 324–334. Springer-Verlag, 2001. Extended version.pdf. 23, 164,196

[11]. 19:195–249, 2000. 4, 14, 15, 20, 37, 46, 50, 58, 70, 89, 90, 91, 92, 95, 110, 112

[12]. N. Th´eriault. "SPA resistant left-to-right integer recodings". In Selected Areas in Cryptography – SAC '05, LNCS, vol. 3897, pp. 345–358. Springer- Verlag, 2006. 23, 128, 133, 164

E.kesavulu Reddy is working as Assistant Professor in Dept. of. Computer Science (MCA) S V University. Tirupati (AP)-India. He is perusing the PhD under the guidance of Prof.P.Govind Rajulu Dept of Computer Science, S .V . University, Tirupati.

ISBN: 978-988-17012-0-6
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)