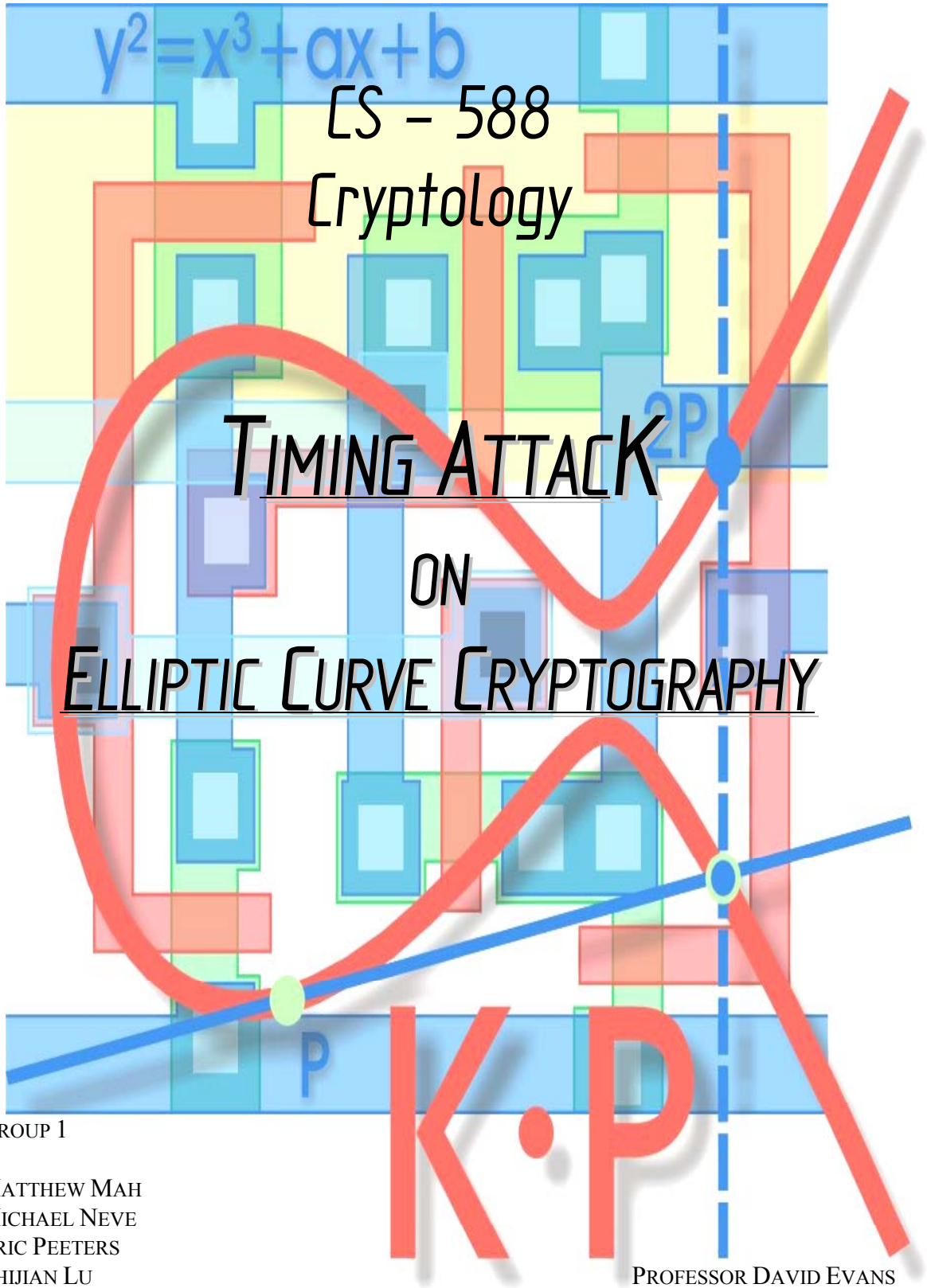


CS – 588 Cryptology

TIMING ATTACK ON ELLIPTIC CURVE CRYPTOGRAPHY

GROUP 1

MATTHEW MAH
MICHAEL NEVE
ERIC PEETERS
ZHIJIAN LU

PROFESSOR DAVID EVANS

Table of Contents

Introduction

As subject for this project, we first planned to focus upon smart card timing attacks. Smart cards are widely used through Western Europe and will probably appear soon in America. They are used in various application fields and with different levels of complexity and security.

Timing attacks attempt to exploit the variations in computational time for private key operations to guess the private key. This type of attack is primitive in the sense that no specialized equipment is needed. An attacker can break a smart card key by simply measuring the computational time required by the card to respond to user inputs and recording those user inputs. The viability of this attack is important to any smart card implementation using vulnerable cryptosystems. An attacker with prolonged passive eavesdropping ability may be able to break the private key and gain access to the information stored on the card. This will give the attacker access to sensitive information or money.

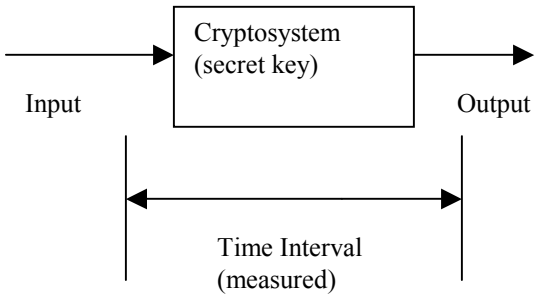
Later – and after readings – we focused deeper: produce a new timing attack. We have glanced through the Internet to find a cryptosystem not yet analyzed for timing weaknesses. Hence, it appears that the vulnerability of Elliptic Curve Cryptology to timing attacks has not been widely studied. We have thought that this subject could be satisfactory and innovative.

This report is subdivided in three parts: we first start talking about the basics of the timing attacks on a RSA implementation; we then develop a brief presentation of Elliptic Curves and EC Cryptology. The last and major part of the report is dedicated to the timing attacks on an open-source implementation of ECC and our diagnosis about this last point.

1. Timing Attacks

Recently, a new class of cryptanalysis aimed at a cryptosytem's implementation-specific weaknesses has attracted great interest. This kind of cryptanalysis exploits the leak of information such as timing, power consumption, and electromagnetic radiation from system operations to facilitate attacks on the cryptosystem. Since the information used by the attack is not the in the "main channel", the input or output, we call these types of attacks "side-channel" attacks. In this paper, we will focus on timing attacks.

Let's think the cryptosystem as a black box with input and output which constitute the "main channel" of the system. We can measure the time it takes for the system to give an output after given an input. The time required for different inputs may vary, forming a timing distribution. If this timing distribution is related to the secret (key bits) in the system, we may have a way to reveal the secret key.



Mathematic Model

Let us denote a set of inputs (plaintexts) to the system by $S_M = \{M_1, M_2, ..., M_n\}$. All the possible keys compose the key set denoted by $S_K = \{K_1, K_2, ..., K_d\}$, where d is the number of possible keys. If the cryptosystem implementation we want to attack is vulnerable to timing attacks, the timing distribution of the input will be dependent on the key used in the system. Thus for key K_i, we will have a timing distribution donated by $P_i(t) = f(S_M, K_i)$, which is different from that of other keys.

For the system we want to attack, we measure the timing information for a set of input values from the set S_M, and form a timing distribution $P(t)$. The attack to the system will be reduced to a usual detection problem which tries to detect K_i knowing $P_i(t)$ and $P(t)$. We can apply, at least in theory, regular detection solutions to solve the problem. For example, the detection problem has a general form of the solution: if $T(P(t), K_i) >$ Threshold(K_i, S_M), K_i is detected.
As long as we find the proper transform function T() and the threshold functions, we break the system.

Timing Attack on RSA

The timing attack on RSA was first proposed by Kocher[1]. In Kocher's paper, a theoretical analysis is given on the timing attack on an RSA implementation based on the following modular exponentiation algorithm:

Let $s_0 = 1$
For k= 0 upto w-1

If (bit k of x) is 1 then

Let $R_k = (s_k * y) \bmod n$ (1)

Else

Let $R_k = s_k$

Let $s_{k+1} = R_k^2 \bmod n$

End For

If we have known exponent bits 0…(b-1), we will know the value of s_b. If bit b is 1, operation (1) will be performed, and for some values of s_b, operation (1) will take longer than for other values of s_b due to modular reduction operations. If we find such a timing difference between these two kinds of value s_b, we know bit b is 1. Otherwise bit b is 0. After we repeat the attack for the entire loop, we will know the entire exponent, which is supposed to be the secret.

Dhen et al[2] gives a practical implementation of the timing attack for the above modular exponentiation algorithm. For simplicity, let's assume we know bit 0…(b-1) and we want to find the bit b of the exponent. We know the value of y, which is open to public, and can calculate the value of s_b from bits 0 to b-1 of the exponent and y. Also we have a large number of inputs to the cryptosystem and record the corresponding timings used in the system. Since we now target bit b of the exponent, we can write the timing for a specific input M (the value of y) as:

$T_M = T_{n1}(M) + T_b(M) + T_{n2}(M)$ +N(M) , T_{n1} is the time spent for iterations before the that for bit b, T_b is the time spent for bit b, and T_{n2} is the time spent after bit b. N is the time for other operations in the cryptosystem plus noise. We will divide the inputs into 4 sets according to the value of s_b:

S10: those inputs whose $(s_b * y)^2 \bmod n$ is done with a reduction

S11: those inputs whose $(s_b * y)^2 \bmod n$ is done without a reduction

S00: those inputs whose $(s_b)^2 \bmod n$ is done with a reduction

S01: those inputs whose $(s_b)^2 \bmod n$ is done without a reduction

Notice that S10 and S11 form a disjoint partition for the entire input set, while S00 and S01 form another disjoint partition for the entire input set. But there will have overlaps between S10 and S00, S10 and S01, S11 and S00, S11 and S01. Let's define the following indicator functions for each input set using the timing data we measured:

$$I_{10} = \frac{\sum T_{S10}}{n(S10)} = \frac{\sum((T_{n1}(S10) + T_{n2}(S10) + N(S10))}{n(S10)} + \frac{\sum T_b(S10)}{n(S10)},$$

$n(S10)$ is the number of elements in the input set S10

Similarly

$$I_{11} = \frac{\sum T_{S11}}{n(S11)} = \frac{\sum((T_{n1}(S11) + T_{n2}(S11) + N(S11))}{n(S11)} + \frac{\sum T_b(S11)}{n(S11)}$$

$$I_{00} = \frac{\sum T_{S00}}{n(S00)} = \frac{\sum((T_{n1}(S00) + T_{n2}(S00) + N(S00))}{n(S00)} + \frac{\sum T_b(S00)}{n(S00)}$$

$$I_{01} = \frac{\sum T_{S01}}{n(S01)} = \frac{\sum((T_{n1}(S01) + T_{n2}(S01) + N(S01))}{n(S01)} + \frac{\sum T_b(S01)}{n(S01)}$$

Since the left part of the indication function is independent of the value of bit b, the probability theory tells us that the left part in all the indication functions will be evaluated very close to the same constant value when the number of elements in our input sets is large enough. So we will have the following conclusions on the indication functions when looking back at the modular exponentiation algorithm:

If bit b of the exponent is 1, the right part in I_{10} will be significantly larger than that in I_{11}, and we will have $I_{10} > I_{11}$. However, since S00 has overlaps with both S10 and S11, and S01 has overlaps with both S10 and S11, there will be no significant difference between I_{00} and I_{01}. On the contrary, if bit b is 0, we will have $I_{00} > I_{01}$ and I_{10} is close to I_{11}.

Consider our mathematic model for the timing attack, we can see that the transform function here is

$$T(P(t), K_i) = \begin{cases} \frac{I_{10}}{I_{11}}, for \quad K_i = 1 \\ \frac{I_{00}}{I_{01}}, for \quad K_i = 0 \end{cases}$$

The threshold function is

$$\text{Threshold}(K_i, S_M) = 1 \text{ for } \begin{matrix} K_i = 1 \\ K_i = 0 \end{matrix}$$

Extension for Timing Attacks

We have briefly introduced the timing attack on an implementation of RSA. There are several points we want to make. Timing attacks, just like other "side-channel" attacks, aim at specific implementations of a cryptosystem, which means the attack may be successful against one implementation of a cipher but not another implementation of the same cipher. Timing attacks have proved to be very powerful. Researchers have found timing attacks on both asymmetric ([1],[2]) and symmetric [3][4] cryptosystem. The underlying ideas of these attacks are the same as that we introduced in above. Furthermore, our timing attack model can be extended to other "side channel" attacks. For example, if we substitute the Hamming-weight information for the timing information in our model, we will have the Hamming-weight attacks proposed in [3].

2. Elliptic Curve Cryptology

This current section introduces the basic concepts of Elliptic Curves. Our purpose here is to present the main ideas we've used in this project to understand the EC, but certainly not to be complete: excellent references are available to provide the whole mathematical context used in ECC. Nevertheless, we have attached importance to the correctness of all the defined operations.

Introduction

Elliptic curves as algebraic/geometric entities have been studied extensively for the past 150 years, and from these studies has emerged a rich and deep theory. Elliptic curve systems as applied to cryptography were first proposed in 1985 independently by Neal Koblitz from the University of Washington, and Victor Miller, who was then at IBM, Yorktown Heights.[5]
These curves have allowed establishment of a new generation of asymmetric cryptographic algorithms. The big win with ECC, as compared to other public-key algorithms, is key size. A fairly typical key size for RSA is 1024 bits--this would take approximately 10^11 MIPs-years to break. A mere 160-bit ECC key offers the same level of security. This advantage only increases with security level--something that will be important as computer power continually grows. A 2048-bit RSA key and a 210-bit ECC key are equivalent.
ECC also has less computational overhead than RSA, primarily because it does not have to analyze prime numbers, a fairly expensive operation.[5]
ECC can be used with SSL scheme, certificates, Diffie-Hellman key agreement, El-Gamal and protocols such ECDSA (Elliptic Curve Digital Signature Algorithm).

This could lead ECC to be a major tool/element of tomorrow's cryptology. While ECC has not been as extensively researched as RSA, to date all research has confirmed ECC to be secure.[5]

Elliptic curve operations

The general equation defining a Elliptic curve is: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$. Some of the a_i parameters may be zero: this could lead to simpler forms of the general equation.
To demonstrate the basic general operations over EC, we will use: $y^2 = x^3 + ax + b$ (e.g. : a = -7 & b = 6).
The shape of this equation is shown in figure 1. The figure is symmetrical about the x-axis.

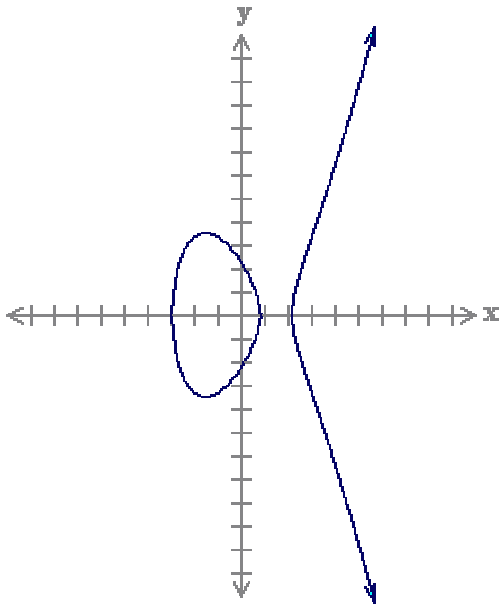


Figure 1 – EC representation

We choose a start point P on the curve.

Addition

We define here the geometric addition of two points of the EC. We take another point Q on the curve. We compute R = P + Q. It is a graphical operation consisting of tracing a line through P and Q. This line defines a unique intersection with the EC: the point –R. R is found by taking the symmetric point with respect to the x-axis (see figure 2).

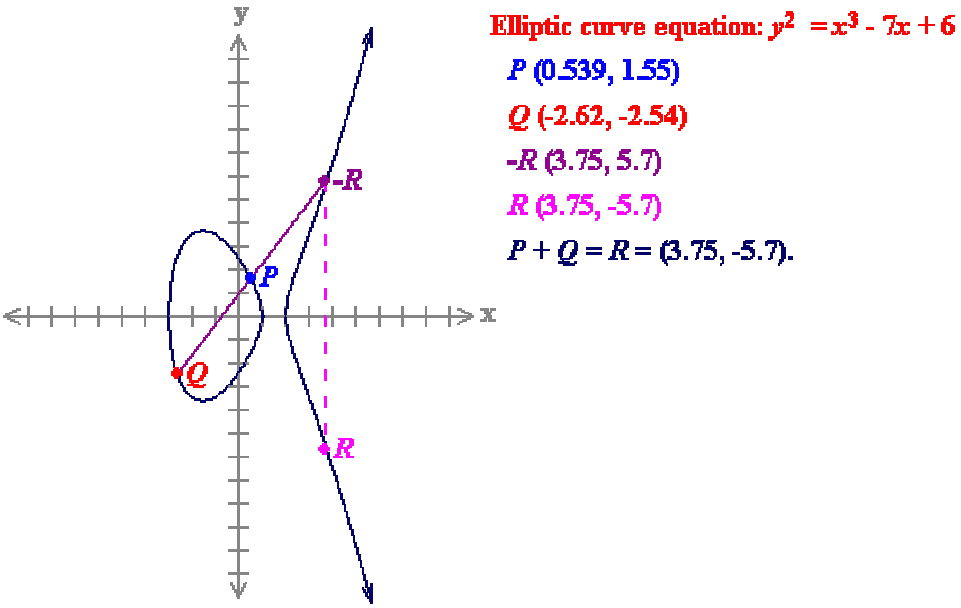


Figure 2 – Example of addition

Formally:
$P = (x_1,y_1)$
$Q = (x_2,y_2)$

$R = P + Q = (x_3,y_3)$

Where,
$x_3 = \theta^2 - x_1 - x_2$
$y_3 = \theta (x_1 + x_3) - y_1$
with $\theta = (y_2 - y_1)/(x_2 - x_1)$.

If P = Q, R = P + Q is equivalent to adding a point to itself: doubling point P.

Doubling

The operation needs a single point and consists of finding a point 2P. We draw a tangent line to the EC at point P. This line intersects the curve in a point –R. Reflecting this point across the x-axis gives R: R = 2P. See figure 3.

Formally: the only difference with an addition is the definition of θ.
$\theta = (3x_1^2 + a)/2y_1$.

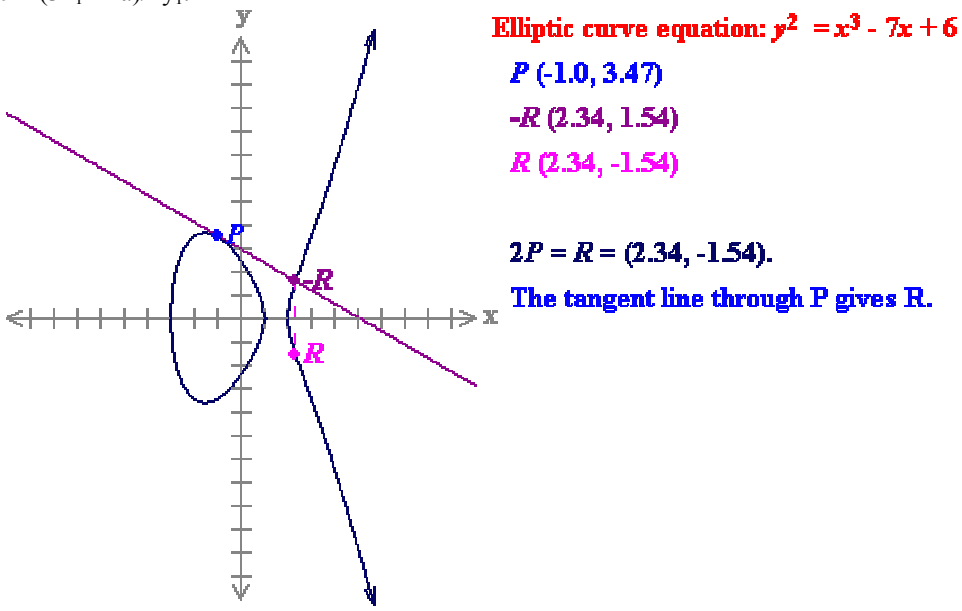


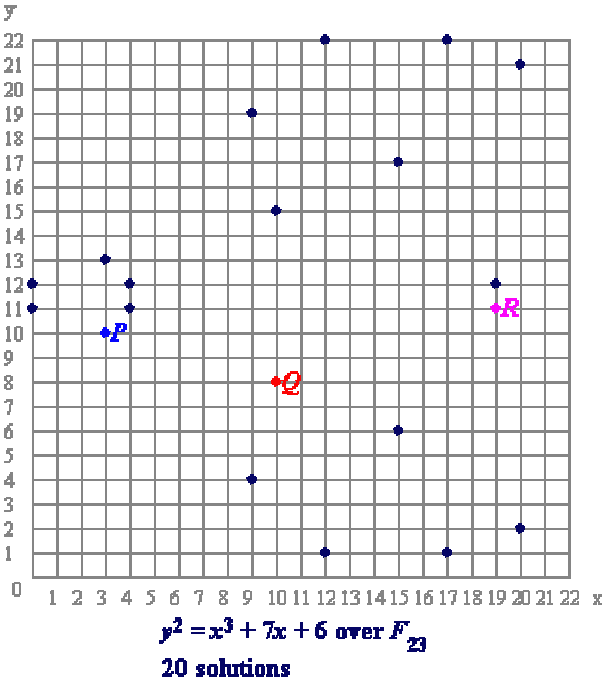
Figure 3 – Example of doubling a point P

Calculations over the real numbers are slow and inaccurate due to round-off error. Cryptographic applications require fast and precise arithmetic; thus elliptic curve groups over the finite fields of F_p and F_{2^m} are used in practice.[5]

EC over prime field

Recall that the field Fp uses the numbers from 0 to p - 1, and computations end by taking the remainder on division by p. For example, in F_{23} the field is composed of integers from 0 to 22, and any operation within this field will result in an integer also between 0 and 22.[5] There are then a finite number of points.

Similar operations as addition or doubling are defined over the field. For example, the EC over F_{23} with a = 7 & b = 6 is shown in figure 4. The addition is computed the same way: two points P and Q are taken, the result gives the point R. The operations of addition and doubling exhibit the property of closure over the elliptic curve; the result of these computations yields another integer point on the EC. This is difficult to prove and we will accept this as a given.



P (3, 10)
Q (10, 8)
R (19, 11)

$l = (y_P - y_Q) * (x_P - x_Q)^{-1} \bmod p$
$= 2 * (-7)^{-1} \bmod 23$
$= 2 * 16^{-1} \bmod 23$
$= 2 * 13 \bmod 23$
$= 3$

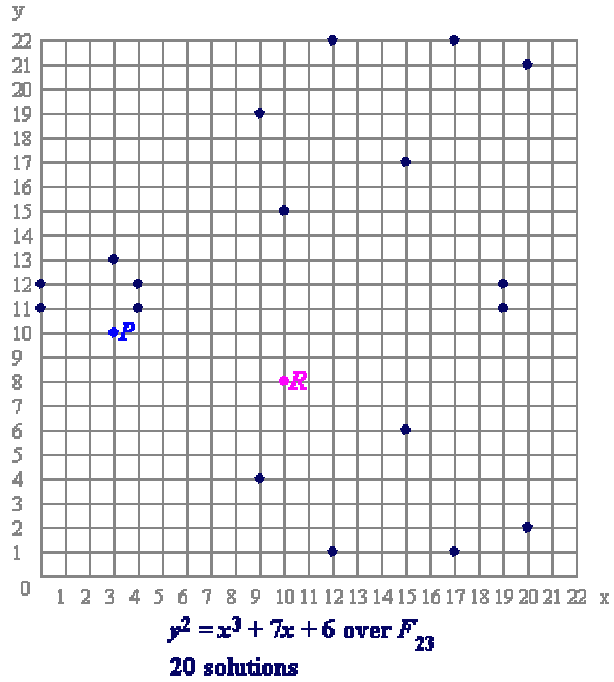
$x_R = l^2 - x_P - x_Q \bmod p$
$= 9 - 3 - 10 \bmod 23$
$= 19$

$y_R = -y_P + l * (x_P - x_R) \bmod p$
$= -10 + 3 * (3 - 19) \bmod 23$
$= 13 + 3 * 7 \bmod 23$
$= 13 + 21 \bmod 23$
$= 11$

$P + Q = R = (19, 11)$.

Figure 4 – Example of addition over F_{23}

The figure 5 shows a doubling of point P over F_{23}.



P (3, 10)
R (10, 8)

$l = (3x_P^2 + a) * (2y_P)^{-1} \bmod p$
$= 34 * 20^{-1} \bmod 23$
$= 11 * 15 \bmod 23$
$= 4$

$x_R = l^2 - 2x_P \bmod p$
$= 16 - 6 \bmod 23$
$= 10$

$y_R = -y_P + l * (x_P - x_R) \bmod p$
$= -10 + 4 * (3 - 10) \bmod 23$
$= 13 + 4 * 16 \bmod 23$
$= 13 + 18 \bmod 23$
$= 8$

$2P = R = (10, 8)$.

Figure 5 – Example of doubling over F_{23}

EC over binary fields

Elements of the field F_{2^m} are m-bit strings. The rules for arithmetic in F_{2^m} can be defined by either polynomial representation or by optimal normal basis representation. Since F_{2^m} operates on bit strings, computers can perform arithmetic in this field very efficiently. An elliptic curve with the underlying field F_{2^m} is formed by choosing the elements a and b within F_{2^m} (the only condition is that b is not 0).
The elliptic curve includes all points (x,y) which satisfy the elliptic curve equation over F_{2^m} (where x and y are elements of F_{2^m}). An elliptic curve group over F_{2^m} consists of the points on the corresponding elliptic curve. There are finitely many points on such an elliptic curve.[5]

Again addition and doubling operations are defined over F_{2^m}. Although these operations are different since the representation of elements are different, the general behavior over F_{2^m} is similar to F_p. Again, these operations exhibit closure and yield only points on the EC.

Here, we will describe one method for multiplying in a binary field, the shift-and-add method described in [6]. This multiplication is a necessary component step in computing the sum of two elliptic curve points or an elliptic curve doubling. Each element in the field F_{2^m} may be represented by a polynomial f(x) of degree <= m-1, with binary coefficients. Reductions in the field are done modulo a fixed irreducible polynomial f(x) of degree m. Operations with these polynomials will yield the proper results. For example, given elements A, B, and C in F_{2^m} with AB = C, multiplying the polynomial representation of A with that of B will yield that of C. This is a property of the Galois field (finite field). We will assume that the polynomial f(x) is written with highest powers first, e.g. $x^4 + x^2 + 1$. Each polynomial may be written as a bit string, where each bit is a coefficient of the polynomial. The polynomial $x^4 + x^2 + 1$ may be written as 0…010101, where the number of leading zeroes depends upon the field order. If we take two polynomials $a(x) = \sum_{i=0}^{m-1} a_i x^i$ and $b(x) = \sum_{j=1}^{m-1} b_j x^j$ we can use the shift-and-add method to find f(x)*g(x). The method is essentially an application of the distributive property to multiply two polynomials, as one learns in math classes. We rewrite $a(x) * b(x) = \sum_{i=0}^{m-1} a_i x^i \sum_{j=0}^{m-1} b_j x^j = \sum_{i=0}^{m-1} a_i \sum_{j=0}^{m-1} b_j x^{i+j}$. It is easy to find $a_i \sum_{j=0}^{m-1} b_j x^{i+j}$ because its bit string is simply the bit string of $\sum_{j=0}^{m-1} b_j x^j$ left shifted by i places and multiplied by 1 or 0. So for the polynomials $(x^4 + x^2 + 1)$ and $(x^3 + x + 1)$, $(x^4 + x^2 + 1) * (x^3 + x + 1) = x^4(x^3 + x + 1) + x^2(x^3 + x + 1) + (x^3 + x + 1)$. The addition step is a simple xor of the two bit strings since each coefficient has a binary value.

The algorithm as presented in [6]:
Input: binary polynomials a(x) and b(x) of degree at most m-1
Output: c(x) = a(x) * b(x) mod f(x), where f(x) is fixed irreducible polynomial of field

1. If $a_0 = 1$ then c = b else c = 0
2. for i from 1 to m-1 do
    a. b = b *x mod f(x) // this step is the left shift
    b. If $a_i == 1$ then c = c + b
3. return c

For this particular algorithm for multiplying field elements, the computation time is dependent upon the number of 1's in the bit string representation of a(x), which is the Hamming weight of a(x). We also know that when deg a(x) + deg b(x) > m-1, reductions must occur mod f(x).

3. El-Gamal scheme with EC

The El-Gamal cipher may be implemented with any group. We describe the cipher using an elliptic curve over a finite field as our group. We assume Bob and Alice agree on a curve over F_p and a point P. They publish the curve and the point.
Bob picks a number "y" randomly between 0 and p-1, he computes Y=yP and publishes his public key Y. The security relies on the difficulty to find Y knowing P: an attacker must compute all steps $X_{i+1} = X_i + P$ ($P \rightarrow 2P \rightarrow 3P \rightarrow \ldots \rightarrow yP$) until X_i equals Y; while Bob can obtain Y by using efficient algorithm (based of doubling operation) to calculate Y rapidly.

Scheme:

Alice Bob

F_q,E
P∈E,Y

m: the message
k: a random number between 0 and p-1

a=kP
G=kY
b=m+G
c=(a,b) ⟶ G'=ya
m'=b-G'=m

Proof:

m' = b-G'
= b-ya
= b-ykP
= b-kY (Y= yP)
= m+G-G
= m

4. Timing Attack on ECC

The timing attack requires us to find a partition of an input message set into groups that require different amounts of computational time depending on the key bits. For a timing attack to work, there must be some predictable variation in computational time dependent upon the input messages.

We expected to be able to exploit variations in computing times for multiplications and inverses to guess the key bits in an implementation the El-Gamal cipher in ECC. These two steps are the most time consuming steps in the process of adding two elliptic curve points or doubling an elliptic curve point. As shown above, the shift-and-add method for multiplication certainly has computational time dependent upon the bit strings multiplied. An implementation of ECC using this algorithm for multiplications should be vulnerable to timing attacks.

Decryption in El-Gamal involves only two steps: G'=ya and m'=b-G'=m. The step m'=b-G'=m is relatively simple; it requires one point addition. Finding G'=ya, however, is similar to modular exponentiation. The term ya is the point a added y times; it requires a series of point doublings and additions in reasonably fast implementations. The following is a section of code for computing kP, where k is an integer and P is a point on the EC.

```
/*  now follow balanced representation and compute kP  */

      bit_count--;
      copy_point(p,r);        /* first bit always set */
      while (bit count > 0)
      {
        edbl(r, &temp, curv);
        bit_count--;
        switch (blncd[bit_count])
        {
           case 1: esum (p, &temp, r, curv);
                   break;
           case -1: esub (&temp, p, r, curv);
                    break;
           case 0: copy_point (&temp, r);
         }
      }
```

The balanced representation of the key reduces the overall number of operations required to compute kP. For a more detailed explanation of the balanced representation, refer to [9]. Clearly here, when the balanced representation bit is 1 or –1, the operation will take longer to perform than for a representation bit of 0, but we must find sets of messages that require different amounts of time to be computed in these cases for an effective timing attack. The edbl, esum, and esub operations all require field multiplies and inverses, so we must examine whether these operations require variable time dependent upon the message inputs.

The implementation of ECC written by Rosing that we examined did not use the shift-and-add algorithm for performing multiplication. The algorithm used is based upon table lookups in a fixed size array. The same lookups and computations are performed for all inputs. Below is short selection from the function to multiply two field elements.

```
void opt_mul(a, b, c)
FIELD2N *a, *b, *c;
{
      INDEX i, j;
      INDEX       k, zero_index, one_index;
```

```
      ELEMENT bit, temp;
      FIELD2N     amatrix[NUMBITS], copyb;

…

/*  main loop has two lookups for every position.  */

      for (j = 1; j<NUMBITS; j++)
      {
            rot_right( &copyb);
            zero_index = Lambda[0][j];
            one_index = Lambda[1][j];
            SUMLOOP (i) c->e[i] ^= copyb.e[i] &
                              (amatrix[zero_index].e[i] ^
amatrix[one_index].e[i]);
      }
}
```

This multiplication of binary field elements will not take variable time. So we must find another step in computing elliptic curve points that results in more variation.

We also found that the computation of inverses was not dependent upon the input. The same table lookups used in the multiplication algorithm are also used in the inverse algorithm.

```
void opt_inv(a, result)
FIELD2N *a, *result;
{
      FIELD2N     shift, temp;
      INDEX m, s, r, rsft;
      int count = 0;

/*  initialize s to lg2_m computed in genlambda.  Since msb is always
set,
      initialize result to input a and skip first math loop.
*/

      s = lg2_m - 1;
      copy( a, result);
      m = NUMBITS - 1;

/*  create window over m and walk up chain of terms  */

      while (s >= 0)
      {
            r = m >> s;
            copy( result, &shift);
            for (rsft = 0; rsft < (r>>1); rsft++) rot_left( &shift);
            opt_mul( result, &shift, &temp);
            if ( r&1 )              /* if window value odd  */
            {
                  rot_left( &temp); /*  do extra square  */
                  opt_mul( &temp, a, result);   /*  and multiply  */
            }
            else copy( &temp, result);
            s--;
      }
}
```

Additionally, the loop parameters are not dependent upon the input. They are dependent upon the size of the field F_{2^m}. The operations performed in the loop itself are also independent of the input. We have seen above that multiplication is independent of input, and the `rot_left` function is as well. So the inverse computation is not dependent upon the input.

Neither the field multiplies nor the inverse computations are dependent upon the message inputs. So we must conclude that the `edbl`, `esum`, and `esub` functions are also not message input dependent and that we cannot determine key bits from timing measurements of the El-Gamal cipher for this implementation.

Attached is a histogram of timing measurements run on a set of approximately 5000 random messages (figure 6). The times for each message are for 15 decryptions of each message to enhance differences in timings. These measurements were taken on a Pentium II 266 mhz processor with 192 MB of RAM. The measurements show much less variation than the samples shown in [1], where a timing attack was successful (figures 7 & 8). This information agrees with our findings that the Rosing implementation of ECC is invulnerable to timing attacks.

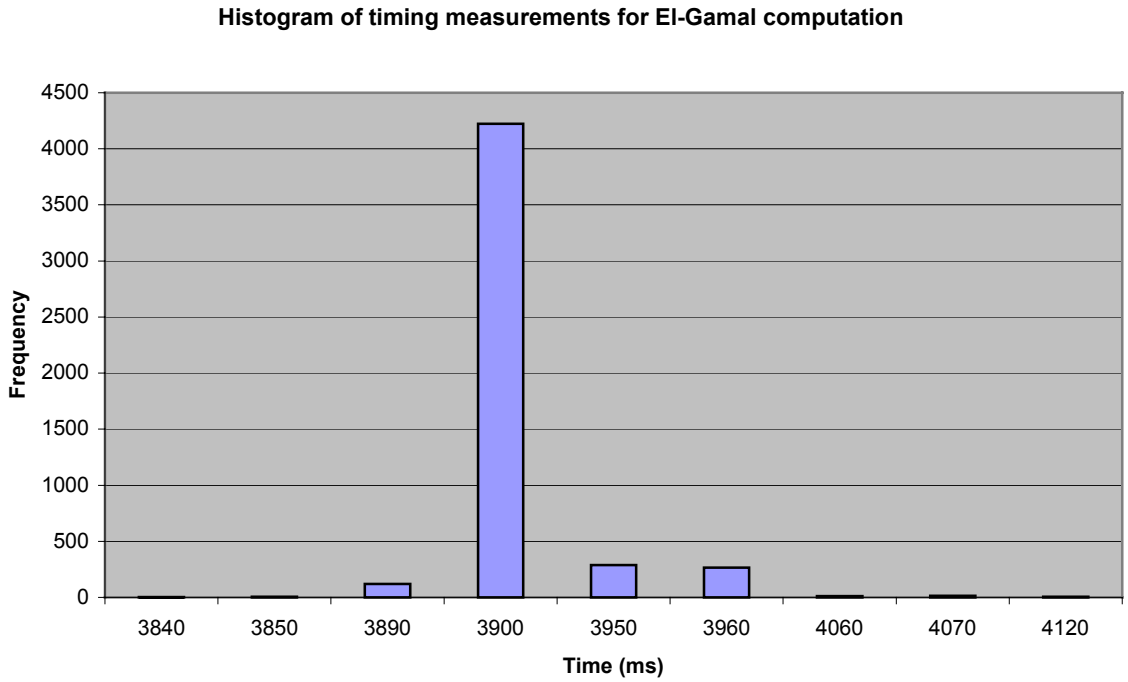
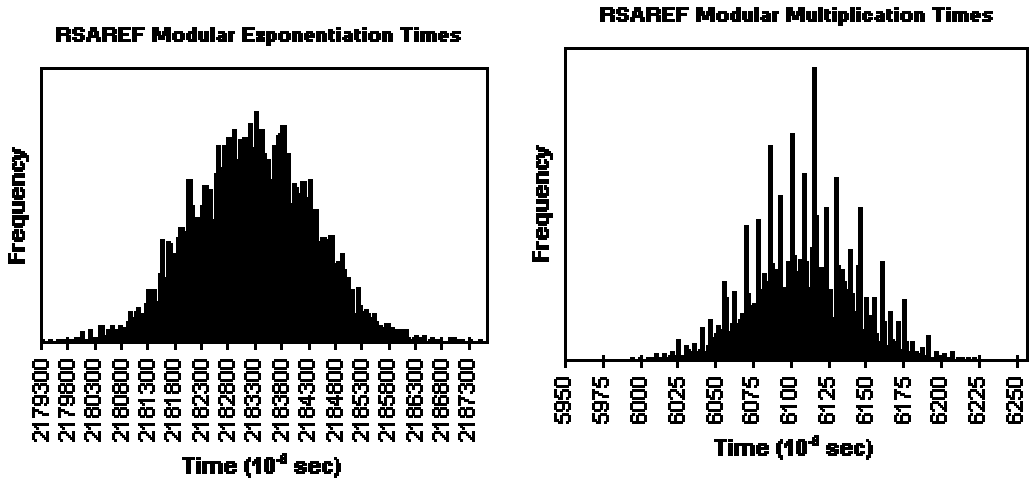


Figure 6 – Histogram of Rosing ECC implementation



Figures 7 & 8 – Histograms of timings [1]

5. Conclusion

With this work, we examined the necessary factors to assess the vulnerability of an ECC system to the Timing Attack. We have tried to find weaknesses in different computational algorithms used in Rosing's ECC implementation. We have also attempted to run the El-Gamal implementation with randomly generated messages to check if different messages needed different time to be decrypted. But, both analytical approaches provide us with evidence that this implementation is resistant to a Timing Attack.

However, we cannot claim so far that all implementations of ECC are resistant to timing attacks. It seems that the basic algorithms used to implement ECC system could cause the system to be vulnerable to a timing attack. The implementation determines whether a cryptosystem is vulnerable to a timing attack.

6. References

[1] P.Kocher,"Timing Attacks on Implementations of Diffe-Hellman, RSA, DSS, and Other Systems," Advances in Cryptology-CRYPTO'96 Proceedings, Springer-Verlag, 1996, pp. 104-113
[2] J.-F. Dhem, F.Koeune, P.-A. Leroux, P.Mestre, J.-J. Quisquater and J.-L. Willems, "A practical implementation of the timing attack," Proc. CARDIS'1998, Smart Card Research and Advanced Applications, LNCS. Springer, 1998
[3] J. Kelsey, B. Schneier, D. Wagner and C. Hall, " Side Channel Cryptanalysis of Product Ciphers" ESORICS 1998: 97-110
[4] F. Koeune and J.-J. Quisquater, "A timing attack against Rijndael" UCL Technical Report CG-1999/1
[5] www.certicom.com
[6] Darrel Hankerson, Julio Lopez Hernandez and Alfred Menezes, "Software Implementation of Elliptic Curve Cryptography Over Binary Fields", Cryptographic Hardware and Embedded Systems, 2000.
[7] Don Johnson and Alfred Menezes, "The Elliptic Curve Digital Signature Algorithm (ECDSA)", 1999.
[8] www.vlsi.informatik.tu-darmstadt.de
[9] Implementing Elliptic Curve Cryptography, Michael Rosing,1998 (http://www.manning.com/Rosing/)