Pertinent Side Channel Attacks on Elliptic Curve Cryptographic Systems

Stanford University
CS259c/MATH250: Elliptic Curves in Cryptography

December 15, 2011

1 Introduction

Elliptic curve cryptosystems have become the most trusted, secure systems in widespread use today. However, the recent advent of utilizing side channel information poses an immediate and powerful threat to systems which are not prepared for this exact type of attack. Side channel analysis (SCA) attacks use some measured quantities outside the scope of the actual cryptographic algorithm, in addition to some knowledge of the algorithm used, in order to break the cryptosystem and obtain private data. Specifically, processing time and consumed power may be measured in order to gain information which is then used to deduce the secret key. Side channel attacks of this kind are non-invasive, and unlike using reverse engineering techniques with a tampered device, will usually go unnoticed by the victim. These attacks fall into three main categories: fault analysis, timing attacks, and power attacks. In the context of elliptic curves, fault analysis involves injecting random register faults while a computation is being done, resulting in a point not on the correct elliptic curve but on another curve which is known by the attacker. This paper will focus on the latter two methods: timing and power attacks.

In the remainder of the paper, a simple algorithm used for point multiplication will be outlined, since this can be used in subsequent examples to illustrate the general attack procedure. Then, timing-based and power-based attack methods will be discussed, along with examples and countermeasures. A detailed examination of differential power analysis will be explained, since this is one of the most powerful and easily implementable side channel attacks currently known. A brief discussion of related side-channel attacks and future possibilities will conclude the paper.

2 Naive double-and-add algorithm

Given an elliptic curve $E : y^2 = x^3 + Ax + B$ defined on finite field $\mathbb{F}_q$, multiplying a point $P \in E(\mathbb{F}_q)$ by a scalar $k \in \mathbb{F}_q$ means to add P to itself k times. When k is expressed in binary digits, this lends itself to a simple double and add algorithm. Let $k = k_{L-1}k_{L-2}...k_1k_0$ be expressed in binary. The algorithm is as follows (∞ corresponds to the point at infinity, an identity element for all elliptic curves):

```
Algorithm 1: Double and Add
1. Q = ∞
2. for i=L−1 to 0:
3.     Q = [2]Q
4.     if $k_i$==1: Q = Q + P
5. output Q
```

The algorithm simply steps through the bits of k and adds P to Q whenever it encounters a set bit. Otherwise, it doubles the value currently in Q, since each step through the binary k corresponds to one higher power of 2. The advantage of this algorithm is that it computes $[k]P$ in only $w(k)$ adds, where $w(k)$ is the Hamming weight of k, the number of 1's in k. A direct algorithm computing $[k]P = \underbrace{P + P + ... + P}_{k\ times}$ takes k adds and is therefore less efficient than double-and-add. However, as will be shown, this naive algorithm contains security flaws which can be exploited using SCA attacks.

3 Timing analysis attacks

3.1 Description, example, countermeasures

Simple timing attacks (STA) rely on the fact that different operations, or different inputs to the same operation, can have a large timing variance. By measuring only the time that computations are taking, an attacker can quite easily deduce private key data, given that he knows the specific computational algorithm being used. For example, if the double-and-add method for point multiplication is being used, an attacker may know that the secret key will be multiplied by some point P at some point in the protocol. In El Gamal encryption, for example, the decryption algorithm involves finding $c_2 - [a]c_1$, where c_1 and c_2 are two publicly known points on the elliptic curve which represent the ciphertext. Depending on the value of a, the time between observed power consumption peaks in the processor can reveal a, without having to know either the value of the power or the ciphertexts. To illustrate this, we follow the example given in [1].

From the double-and-add algorithm, it is clear that when a bit of k is 0, the point Q is doubled, and when a bit of k is 1, the point Q is doubled and added to P. Since point doubling ($P + P$) requires slightly fewer processor arithmetic operations than arbitrary point addition ($P + Q$, $P \neq Q$) due to the ability to reduce the group law for $P + P$, the attacker can distinguish between these two cases, and the key can be reconstructed. This was one very concrete, simple example of how practical it is to attack an FPGA implementation with nothing more than an oscilloscope, although the countermeasures are easily implemented. The easiest way to combat this type of attack is to insert dummy adds after step 4 of the algorithm, and to store it in some ignored variable such as R. Then, regardless of each bit of k, the same number of processor operations will be performed, rendering simple timing analysis useless. Similarly, one could use the full group law for arbitrary point addition even when doing the point doubling.

4 Power analysis attacks

4.1 Simple power analysis and countermeasures

Power analysis attacks are the next step up in the arsenal of side channel attacks, and are very similar to timing attacks. In simple power attacks (SPA), the power or current trace of a microprocessor is measured. Since the current drawn by the processor is different for different types of operations (or different inputs), the secret key may again be recovered. Contrary to simple timing analysis attacks, the actual shape of the signal must be analyzed (as opposed to only the time spacing of peaks), which could require some basic signal processing techniques (i.e. noise variance characterization and reduction) in addition to visual inspection. It has been shown [2] that in some symmetric encryption schemes such as DES, and some public schemes such as RSA, this type of attack is feasible given a processor unequipped with the proper countermeasures. For example, the initial permutation and 16 DES rounds (involving permutations, transformations via lookup table, and bit operations) are clearly visible in the power consumption trace from a typical smart card (Figure 1).

Countermeasures include the ones taken for simple timing analysis attacks in addition to multiplication algorithms such as the Montgomery Power Ladder, which, for every bit of k, stores both an addition and a doubling. Since there are no conditional branches, the algorithm doesn't depend on the actual bits of d and is thus SPA-secure. This method is especially nice since it doesn't even reveal the Hamming weight (number of 1's) of the key, as opposed to other blinding techniques which might. Algorithm 2 below outlines this method.

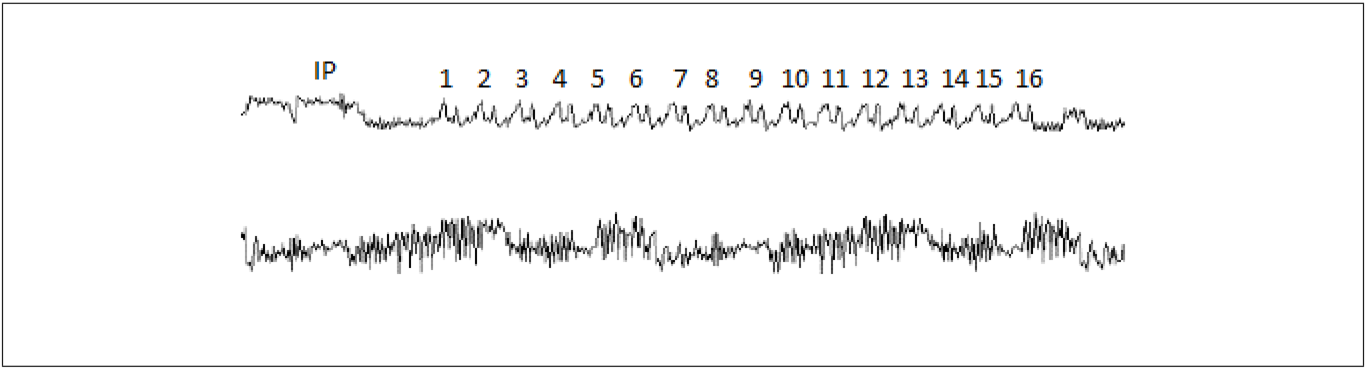

Figure 1: From [2]. Top: Simple power trace of a DES implementation where the initial permutation (IP) and 16 main rounds are clearly visible. Bottom: Zoomed-in portion of the IP, showing the high resolution and amount of data contained in the trace.

```
Algorithm 2: Montgomery Ladder Multiplication
1. Q[0] = P
2. for i=L−2 to 0:
3.    Q[0] = 2Q[0]
4.    Q[1] = Q[0] + P
5.    Q[0] = Q[$d_i$]
6. output Q[0]
```

The Montgomery exponent ladder method for point multiplication is resistant to most timing and power SCAs, but might be vulnerable to more sophisticated methods of attack, as shown in the next section.

4.2 Differential power analysis

4.2.1 Description and example

A much more powerful version of power-based attacks is differential power analysis (DPA), which uncovers statistical relationships between correct and incorrect guesses at a hypothesized bit of the secret key. The following description and example of DPA is adapted from [5]. Whereas SPA looks at one power trace from which all information is deduced, DPA looks at *many* power traces and analyzes them statistically. To uncover these statistical relationships requires some signal processing on the power trace data, but yields correlations that will go undetected in simple power analysis. The two key assumptions for differential power analysis is that the algorithms used by the processor must be known ahead of time, and that the secret key throughout the entire process must be kept fixed. For this reason, signature schemes such as ECDSA are not vulnerable to such an attack (since the secret changes from message to message), while general elliptic curve encryption schemes (such as El Gamal) and key exchange protocols (such as elliptic curve Diffie-Hellman) are.

Differential power analysis consists of two distinct phases: data measurement, and signal processing. In the first phase, the side channel is measured by sampling power consumption during N intermediate steps of some computational algorithm. Then, by computing the intermediate values of the algorithm under some guessed key value (or a portion of the key value), we are able to begin the second phase, which involves computing the correlations between each set of hypothetical intermediate values and the power measurements (traces). The key result of this technique is that naive implementations of elliptic curve cryptosystems can be compromised in relatively few power trace measurements (on the order of 1000) [5].

This process is best illustrated with an example. Say we are using El Gamal encryption, and our task is to find d, given $P \in E(\mathbb{K})$ and $Q = [d]P \in E(\mathbb{K})$ for an elliptic curve E over some finite field $\mathbb{K}$. If we were to approach this problem from a purely mathematical standpoint, this problem boils down to the discrete logarithm problem, which, in a field of large enough order, is computationally infeasible. Now assume further that we know that the processor uses the Montgomery Ladder (Algorithm 2) to compute point multiplication, so STA and SPA will do us no good.

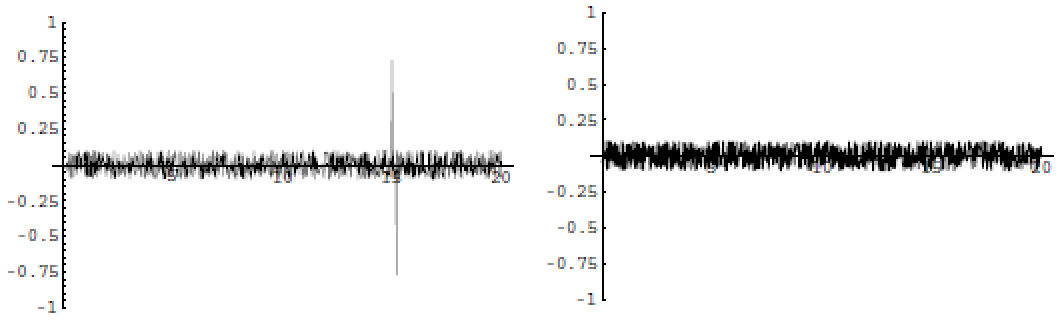

Figure 2: From [5]. Correlated (a) and uncorrelated (b) differential power traces

When the processor computes $[d]P$ in order to decrypt, we can do the following:

1. We observe distinct points P_i as inputs to the algorithm and measure power consumption traces during the computation of $[d]P_1, [d]P_2, ..., [d]P_k$ resulting in k different power traces $C_i(t), 1 \leq i \leq k$.

2. We observe that if the bit $d_{L-2} = 0$, then $[4]P$ is computed at the next iteration, but if $d_{L-2} = 1$, $[4]P$ is never computed (although $[6]P$ is). We conclude that there will be a correlation between an arbitrary bit of $[4]P$ (we can compute $4[P]$ as a pre-processing step), and our power trace. This is because hardware operations on set bits inherently consume more power than operations on unset bits.

3. We then compute the correlation function $g(t)$ between an arbitrary bit s_i of $[4]\mathrm{P}_i$ (for $1 \leq i \leq k$) and our power traces $C_i(t)$:

$$g(t) = \langle C_i(t) \rangle_{s_i=1} - \langle C_i(t) \rangle_{s_i=0} \ , \tag{1}$$

   where $\langle C_i(t) \rangle$ denotes the average over all C_i at time t. $g(t)$ is just the difference between the average of all power traces where our arbitrary bit is 1, and the average of all power traces where our arbitrary bit is 0. Using the observation (2), if we see a peak in $g(t)$, it will occur at some time t' (the instant that $4P_i$ is computed by the algorithm) since by our observation, s_i is correlated to the power trace if $4P_i$ is computed. In this case, we conclude that $d_{L-2} = 0$. If we see no peak in $g(t)$, then we conclude that $4P_i$ is never computed and that our arbitrary bit has no relation to what has been computed in the processor: thus $d_{L-2} = 1$.

4. We repeat this process, one bit at a time, (i.e. for bit d_{L-3} we compute the correlation between power traces $C_i(t)$ and an arbitrary bit of *either* $[8]P = 2 * (4P)$ if the last key bit was 1 or $[12]P = 2 * (6P)$ if the last key bit was 0), and deduce the entire secret key d.

In summary, we have used knowledge of the algorithm in order to find correlations in the power traces, based on whether or not a certain power of 2 multiple of P is computed. Figure 2 shows the difference between observing a positive correlation between power trace and a bit of $4P_i$, and observing no correlation. Although the multiplication algorithm we exploit is not vulnerable to simple timing or power attacks, we have leveraged some basic signal processing to deduce the secret key d, and we have done it in $\mathcal{O}(L)$ time after taking the measurements. Not only El Gamal cryptosystems are vulnerable to this DPA attack: any decryption algorithm involving multiplication of a fixed secret key with a known point, such as Diffie-Hellman key exchange, is open to this exact type of attack.

Another method of finding correlations in noisy power traces is the Pearson product-moment coefficient r described in detail in [8], which is defined as follows:

$$r = \frac{\sum_{i=1}^{k}(C_i(t) - \langle C_i \rangle)(L_i(t) - \langle L_i \rangle)}{\sqrt{\sum_{i}^{k}(C_i(t) - \langle C_i \rangle)^2 \sum_{i}^{k}(L_i(t) - \langle L_i \rangle)^2}} \tag{2}$$

$C_i(t)$ are the power traces measured in step 1, and $L_i(t)$ are the "leakage" values for input P_i which is somehow deterministically calculated, based on the expected value of some intermediate arbitrary bits of $Q[0]$. This equation is simply $SP/\sqrt{SS_C SS_L}$, where SP is the sum of products, SS_C is the sum of squares of C, and SS_L is the sum of squares of L, where C and L are both mean-adjusted. By looking at individual correlations between samples of the power traces, this metric more precisely finds statistical relationships within the data, although it is a more computationally intensive method. A major benefit of this method is that multiple bits of the key may be guessed at the same time, as opposed to one bit at a time, since the leakage model can take into account more than one bit (in the first DPA example, the leakage value was basically either 0 or 1).

4.2.2 Countermeasures

Three easily-implemented ways to thwart DPA have been presented by [5].

1. *Randomize the secret key d at the beginning of the algorithm.*
   (a) Pick a random k of some length (usually $\approx$ 20 bits)
   (b) Compute $d' = d + k \cdot \#E(\mathbb{K})$ where $\#E(\mathbb{K})$ is the number of points on $E(\mathbb{K})$.
   (c) Use the Montgomery Ladder algorithm to compute $Q = d'P$

   This works because $\#E(\mathbb{K}) \cdot P = \infty$ since the order of a point must divide the order of the group, and therefore $Q = dP + \infty = dP$ as required. Without operating on the fixed secret key d, the algorithm reveals no information about the bits of d and only operates on the random d'.

2. *Blind the point P by a secret point R such that $S = [d]R$ for some known S, known only by processor.*
   (a) Use some multiplication algorithm to compute $d(R + P) - S = dR + dP - dR = dP$.
   (b) Before starting the algorithm again for another multiplication, refresh R (thereby refreshing S) by $R \leftarrow (-1)^b 2R$ for $b \overset{R}{\longleftarrow} \{0, 1\}$

   Since the point $(R+P)$ is random and unknown even when given P, the attacker will not be able to compute the theoretical intermediate values in the DPA (i.e. in the example of the previous section, it would be impossible to compute $[4](R + P)$, $[8](R + P)$, etc.)

3. *Use randomized projective coordinates.*
   (a) Store points $P = (x, y)$ as $P' = (\lambda x, \lambda y, \lambda)$ for some random $\lambda \in \mathbb{K} - \{0\}$ before each doubling and addition of the multiplication algorithm.
   (b) Perform the usual point multiplication to find $Q' = aP' = (x', y', z')$ using the general group law.
   (c) Output $Q = (\frac{x'}{z'}, \frac{y'}{z'})$.

   Now, the attacker cannot determine theoretical intermediate values of the algorithm, and the DPA fails. Similarly, randomized Jacobian coordinates may be used, in which (x, y) is represented as $(x', y', z') = (\theta^2 x, \theta^3 y, \theta)$ for some random $\theta \in \mathbb{K} - \{0\}$. In this case, we would output $Q = (\frac{x'}{z'^2}, \frac{y'}{z'^3})$.

4.2.3 Refined attacks

One interesting related attack is the *refined power attack* or RPA, which is basically a DPA technique focusing on special points P_0 where either the x-coordinate or y-coordinate of P_0. As shown in [7], a point with x-coordinate equal to 0 exists on a curve $E : y^2 = x^3 + Ax + B$ defined on a finite field of prime order $\mathbb{F}_P$, $p > 3$ if and only if B is a quadratic residue modulo p, i.e. $\exists\, r \in \mathbb{F}_P : r^2 \equiv B \ (mod\ p)$. This arises directly from setting $x = 0$ in the equation for the curve. Similarly, a point with y-coordinate equal to 0 exists on the curve E if and only if $x^3 + Ax + B$ has at least one root in $\mathbb{F}_p$. By waiting for a special point P_0 to go into the multiplication algorithm, and then leveraging several computational shortcuts and properties, these refined attacks render the three countermeasures in the previous section useless. For example, if $x = 0$ or $y = 0$ in normal coordinates, then they will still be zero in projective coordinates, and other more sophisticated countermeasures must be taken.

4.3 Template attacks

Very similar to differential power attacks are template attacks [4]. Both techniques consist of two rounds: data collection and data processing (now called “template building” and “template matching”, respectively). The main difference is that now, the attacker is assumed to have full control of the processor: he can feed in an arbitrary number of (key, data) pairs and then sample the resulting power trace N times. By measuring the power consumption trace for one of these (key, data) pairs (k_i, d_i), he can then fully characterize the device, which he then models as an N-dimensional normal random variable:

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^N \cdot det(\mathbf{C})}} \cdot exp(-\frac{1}{2} \cdot (\mathbf{x} - \mathbf{m})' \cdot \mathbf{C}^{-1} \cdot (\mathbf{x} - \mathbf{m})). \tag{3}$$

$\mathbf{C}$ is the NxN covariance matrix which characterizes the degree of dependence between observations x_i and x_j, and $\mathbf{m}$ is an N-vector of mean values, E[$\mathbf{x}$]. As $\mathbf{C}$ and $\mathbf{m}$ fully characterize the power trace we measured, the pair $(\mathbf{m}, \mathbf{C})_{d_i,k_i}$ is called the *template* for input (k_i, d_i). By repeating this process for different (key, data) pairs, we obtain many templates.

Then, all that is left to do is to observe and sample the power trace for an unknown (key, data) pair, resulting in power samples $\mathbf{t}$. By finding the probabilities of observing the trace $\mathbf{t}$ in each of the constructed templates, we can find the most probable template corresponding to the correct (key, data) pair (k^*, d^*):

$$(k^*, d^*) = \arg\max_{k_i,d_i} p(\mathbf{t}|(\mathbf{m}, \mathbf{C})_{d_i,k_i}). \tag{4}$$

By finding the maximum likelihood template, we are basically finding the template which ’fits’ the observed unknown power trace the best, thus revealing to us a good candidate for the secret key. There are few known countermeasures to this type of attack, with one being randomized projective coordinates as with defending from DPA attacks. Although relatively unstudied outside of [4], template attacks could pose a heavy threat if proven feasible.

5 Summary and future

A few of the more pertinent examples of side channel attacks have been presented here. It appears that all that is necessary to protect one’s system against such attacks is knowledge of the existence of SCA, since in most cases countermeasures are very simple to implement. The main threats remaining in the field of side channel attacks seem to be RPA and template attacks, both of which are relatively unexplored areas of research. Additionally, [6] and others have proposed an extension of differential power attacks to measure electromagnetic radiation instead of electrical power. Operating on the frequency-domain signal as opposed to the time-domain signal would require most of the same countermeasures as ordinary DPA with the added benefit (to the attacker) of not needing any physical connection with the device. If these potential methods of attack prove to be feasible, more advanced protection schemes will have to be devised to maintain the level of security offered by elliptic curve cryptography.

References

[1] S. Kadir, A. Sasongko, et al, *Simple Power Analysis Attack against ECC Processor on FPGA Implementation.* 2011 International Conference on Electrical Engineering and Informatics, 17-19 July 2011.

[2] P. Kocher, J. Jaffe, B. Jun, *Introduction to Differential Power Analysis and Related Attacks.* Cryptography Research, Inc, 1998.

[3] H. Cohen, G. Frey, et al, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, chapter 28. Chapman and Hall/CRC, 2005.

[4] M. Medwed and E. Oswald, *Template Attacks on EDCSA*. Information Security Applications, WISA, vol. 5379, 2008, pp. 14-27, 2008.

[5] J.S. Coron, *Resistance Against Differential Power Analysis for Elliptic Curve Cryptosystems.* Cryptographic Hardware and Embedded Systems, vol. 1717 of Lecture Notes in Computer Science, pp. 292-302, 1999.

[6] J. Fan, X. Guo, et al., *State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures.* IEEE, 2010.

[7] L. Goubin, *A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems*, CP8 Crypto Lab, Schlimberg-erSema, pp. 207-208, 2003.

[8] L.C. Brown, P.M. Berthouex, et al, *Statistics for Environmental Engineers, 2nd Ed.*, chapter 31. CRC Press, 2002.