Introduction to Side Channel Attacks

White Paper

Discretix Technologies Ltd.

Author: Hagai Bar-El
Title: Information Security Analyst
Email: hagai.bar-el@discretix.com
Tel: +972-9-8858810
www.discretix.com

ABOUT THIS DOCUMENT

The purpose of this document is to introduce Side-Channel attacks, as well as to assist in the decision making of how to protect cryptographic modules against such attacks.

The document is divided into two parts: The first part presents Side-Channel attacks and provides introductory information about such attacks. The second part presents known methods for protection against such attacks with a brief effectiveness assessment, if such is available.

This document is mainly intended for people who are considering the use of cryptographic modules and who need to compare several options with respect to their security.

INTRODUCTION TO SIDE CHANNEL ATTACKS

"*Side channel attacks*" are attacks that are based on "*Side Channel Information*". Side channel information is information that can be retrieved from the encryption device that is neither the plaintext to be encrypted nor the ciphertext resulting from the encryption process.

In the past, an encryption device was perceived as a unit that receives plaintext input and produces ciphertext output and vice-versa. Attacks were therefore based on either knowing the ciphertext (such as *ciphertext-only attacks*), or knowing both (such as *known plaintext attacks*) or on the ability to define what plaintext is to be encrypted and then seeing the results of the encryption (known as *chosen plaintext attacks*). Today, it is known that encryption devices have additional output and often additional inputs which are not the plaintext or the ciphertext. Encryption devices produce timing information (information about the time that operations take) that is easily measurable, radiation of various sorts, power consumption statistics (that can be easily measured as well), and more. Often the encryption device also has additional "unintentional" inputs such as voltage that can be modified to cause predictable outcomes. Side channel attacks make use of some or all of this information, along with other (known) cryptanalytic techniques, to recover the key the device is using.

Side channel analysis techniques are of concern because the attacks can be mounted quickly and can sometimes be implemented using readily available hardware costing from only a few hundred dollars to thousands of dollars. The amount of time required for the attack and analysis depends on the type of attack (Differential Power Analysis, Simple Power Analysis, Timing, etc.) According to [1], SPA attacks on smartcards typically take a few seconds per card, while DPA attacks can take several hours.

In a general, with a somewhat academic perspective as presented in [7], we may consider the entire internal state of the block cipher to be all the intermediate results and values that are never included in the output in normal operations. For example, DES has 16 rounds; we can consider the intermediate states, *state[1::15]*, after each round except the last as a secret internal state. Side channels typically give information about these internal states, or about the operations used in the transition of this internal state from one round to another. The type of side-channel will, of course, determine what information is available to the attacker about these states. The attacks typically work by finding some

information about the internal state of the cipher, which can be learned both by guessing part of the key and checking the value directly, and additionally by some statistical property of the cipher that makes that checkable value slightly nonrandom.

This document will relate only to the most common types of Side Channel information, which are: Timing attacks, Simple and Differential Power Analysis Attacks and Fault Attacks.

TIMING ATTACKS

Timing attacks are based on measuring the time it takes for a unit to perform operations. This information can lead to information about the secret keys. For example: By carefully measuring the amount of time required to perform private key operations, an attacker might find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems [2]. If a unit is vulnerable, the attack is computationally simple and often requires only known ciphertext.

Cryptosystems often take slightly different amounts of time to process different inputs. Reasons include performance optimizations to bypass unnecessary operations, branching and conditional statements, RAM cache hits, processor instructions (such as multiplication and division) that run in non-fixed time, and a wide variety of other causes. Performance characteristics typically depend on both the encryption key and the input data (e.g., plaintext or ciphertext). Intuition might suggest that unintentional timing characteristics would only reveal a small amount of information from a cryptosystem. However, as shown in [2], attacks exist which can exploit timing measurements, from vulnerable systems, to find the entire secret key.

Timing measurements are fed into a statistical model that can provide the guessed key bit with some degree of certainty (by checking correlations between time measurements).

Computing the variances is easy and provides a good way to identify correct exponent bit guesses. The number of samples needed to gain enough information to allow the recovery of the key are determined by the properties of the signal and the noise. The more noise there is, the more samples will be required. Generally, error correction techniques increase the memory and processing requirements for the attack, but can greatly reduce the number of samples required. [2]

The specific examples that follow present specific background information for timing attacks against operations that are related to asymmetric encryption. Yet, it must be remembered that timing attacks can potentially be used against other cryptosystems, including symmetric functions.

Cryptanalysis of a Simple Modular Exponentiator

Diffie-Hellman and RSA operations consist of computing $R=y^x \ mod \ n$, where n is public and y can be found by an eavesdropper. The attacker's goal is to find x, the secret key. For the attack, the victim must compute $y^x \ mod \ n$ for several values of y, where y, n, and the computation time are known to the attacker and x stays the same. Statistical methods will lead to the recovery of the key from

these measurements. The necessary information and timing measurements might be obtained by passively eavesdropping on an interactive protocol, since an attacker could record the messages received by the target and measure the amount of time taken to respond to each *y*. The attack assumes that the attacker knows the design of the target system, although in practice this could probably be inferred from timing information. The attack can be tailored to work with virtually any implementation that does not run in fixed time.

The complete details about this attack, including the statistical models used, are available in [2].

Montgomery Multiplication and the CRT

Modular reduction steps usually cause most of the timing variation in a modular multiplication operation. Montgomery multiplication eliminates the *mod(n)* reduction steps and, as a result, tends to reduce the size of the timing characteristics. [2]

The Chinese Remainder Theorem (CRT) is also often used to optimize RSA private key operations. With CRT, *(y mod p)* and *(y mod q)* are computed first, where *y* is the message. These initial modular reduction steps can be vulnerable to timing attacks. The simplest such attack is to choose values of *y* that are close to *p* or to *q*, then use timing measurements to determine whether the guessed value is larger or smaller than the actual value of *p* or *q*. If *y* is less than *p*, computing *y mod p* has no effect, while if *y* is larger than *p*, it will be necessary to subtract *p* from *y* at least once. The specific timing characteristics depend on the implementation. [2]

In some cases it may be possible to improve the Chinese Remainder Theorem RSA attack to use known (not chosen) ciphertexts, reducing the number of messages required and making it possible to attack RSA digital signatures [2]. Modular reduction is done by subtracting multiples of the modulus, and exploitable timing variations can be caused by variations in the number of compare-and-subtract steps.

According to [2], it is not yet known whether timing attacks can be adapted to directly attack the *mod p* and *mod q* modular exponentiations performed with the Chinese Remainder Theorem.

Timing Cryptanalysis of DSS

If the modular reduction function runs in non-fixed time, the overall signature time should be correlated with the time for the $(x \cdot r \bmod q)$ computation. The attacker can calculate and compensate for the time required to compute *H(m)*. Since *H(m)* is of approximately the same size as *q*, its addition has little effect on the reduction time. The most significant bits of $x \cdot r$ are typically the first used in the modular reduction. These depend on *r*, which is known, and the most significant bits of the secret value *x*. There would thus be a correlation between values of the upper bits of *x* and the total time for the modular reduction. By looking for the strongest probabilities over the samples, the attacker would try to identify the upper bits of *x*. As more upper bits of *x* become known, more of $x \cdot r$ becomes known, allowing the attacker to proceed through more iterations of the modular reduction loop to attack new bits of *x*. [2]

POWER CONSUMPTION ATTACKS

These attacks are based on analyzing the power consumption of the unit while it performs the encryption operation. By either simple or differential analysis of the power the unit consumes, an attacker can learn about the processes that are occurring inside the unit and gain some information that, when combined with other cryptanalysis techniques, can assist in the recovery of the secret key.

As described clearly in [1], integrated circuits are built out of individual transistors, which act as voltage-controlled switches. Current flows across the transistor substrate when charge is applied to (or removed from) the gate. This current then delivers charge to the gates of other transistors, interconnect wires, and other circuit loads. The motion of electric charge consumes power and produces electromagnetic radiation, both of which are externally detectable.

To measure a circuit's power consumption, a small (e.g., 50 ohm) resistor is inserted in series with the power or ground input. The voltage difference across the resistor divided by the resistance yields the current. Well-equipped electronics labs have equipment that can digitally sample voltage differences at extraordinarily high rates (over 1GHz) with excellent accuracy (less than 1% error). Devices capable of sampling at 20MHz or faster and transferring the data to a PC can be bought for less than $400. [4]

Simple Power Analysis (SPA) Attacks

Simple Power Analysis is generally based on looking at the visual representation of the power consumption of a unit while an encryption operation is being performed. Simple Power Analysis is a technique that involves direct interpretation of power consumption measurements collected during cryptographic operations. SPA can yield information about a device's operation as well as key material.

The attacker directly observes a system's power consumption. The amount of power consumed varies depending on the microprocessor instruction performed. Large features such as DES rounds, RSA operations, etc. may be identified, since the operations performed by the microprocessor vary significantly during different parts of these operations. SPA analysis can, for example, be used to break RSA implementations by revealing differences between multiplication and squaring operations. Similarly, many DES implementations have visible differences within permutations and shifts, and can thus be broken using SPA. [1]

Because SPA can reveal the sequence of instructions executed, it can be used to break cryptographic implementations in which the execution path depends on the data being processed, such as: DES key schedule, DES permutations, Comparisons, Multipliers and Exponentiators.

Most cryptographic units, that were tested, were found to be vulnerable to SPA attacks, though according to [1] it is not difficult to design a system that will not be vulnerable to such attacks. See the next chapter for possible solutions.

Differential Power Analysis (DPA) Attacks

Differential Power Analysis Attacks are harder to prevent. They consist not only of visual but also statistical analysis and error-correction statistical methods, to obtain information about the keys. DPA usually consists of data collection and data analysis stages that make extensive use of statistical functions for noise filtering as well as for gaining additional information about the processes that the unit is performing.

In addition to large-scale power variations due to the instruction sequence, there are effects correlated to data values being manipulated. These variations tend to be smaller and are sometimes overshadowed by measurement errors and other noise. In such cases, it is still possible to break the system using statistical functions tailored to the target algorithm. Because DPA automatically locates correlated regions in a device's power consumption, the attack can be automated and little or no information about the target implementation is required.

In general terms, to implement a DPA attack, an attacker first observes *m* encryption operations and captures power traces *T[1::m][1::k]* containing *k* samples each. In addition, the attacker records the ciphertexts *C[1::m]*. No knowledge of the plaintext is required. DPA analysis uses power consumption measurements and statistical methods to determine whether a key block guess *K* is correct. [4]. Analyzing an outer DES operation first, using the resulting key to decrypt the ciphertexts, and attacking the next DES subkey can find Triple-DES keys. DPA can use known plaintext or known ciphertext and find encryption or decryption keys. [4]

Several improvements can be applied to the data collection and DPA analysis processes to reduce the number of samples required or to circumvent counter-measures. For example, it is helpful to apply corrections for the measurement variance, yielding the significance of the variations instead of their magnitude. One variant of this approach, *automated template DPA*, can find DES keys using fewer than 15 traces from most smart cards. [4]

With regards to asymmetric ciphers, as also explained in detail in [4], public key algorithms can be analyzed using DPA by correlating candidate values for computation intermediates with power consumption measurements. For modular exponentiation operations, it is possible to test exponent bit guesses by testing whether predicted intermediate values are correlated to the actual computation. Chinese Remainder Theorem RSA implementations can also be analyzed, for example, by defining selection functions over the CRT reduction or recombination processes. In general, signals leaking during asymmetric operations tend to be much stronger than those from many symmetric algorithms because of the relatively high computational complexity of multiplication operations.

High-Order DPA (HO-DPA) involves looking at power consumptions between several sub-operations of the encryption operation (and not just on the operation in general). Also, while the DPA techniques described above analyze information across a single event between samples, high-order DPA may be used to correlate information between multiple cryptographic sub-operations. Worth mentioning is that there is no known unit that is vulnerable to HO-DPA and that is not vulnerable to DPA as well. Yet, what is done to prevent DPA must also work against HO-DPA. In other words, the precautions that are taken to prevent DPA should be ones that work against HO-DPA as well, although according to [1] no systems are currently known that are resistant to DPA and are not resistant to HO-DPA.

Differential Fault Analysis (DFA) Attacks

Fault analysis relates to the ability to investigate ciphers and extract keys by generating faults in a system that is in the possession of the attacker, or by natural faults that occur. Faults are most often caused by changing the voltage, tampering with the clock, or by applying radiation of various types.

The attacks are based on encrypting the same piece of data (which is not necessarily known to the attacker) twice and comparing the results. A one-bit difference indicates a fault in one of the operations. Now, a short computation can be applied for DES, for example, to identify the round in which the error has occurred. A whole set of operations can now be carried out (detailed in [3]) to recover one DES sub-key which is the sub-key of the last round. When this sub-key is known, the attacker can either guess the missing 8 bits (the last sub-key uses 48 bits) for which there are only 256 options, or simply peel off the last round for which he knows the sub-key and perform the attack on the reduced DES. This second method can be used also against Triple-DES. The attack is detailed in [3].

Often, DFA can be combined with other attacks as differential-key attacks or differential related key cryptanalysis.

Another type of Fault Analysis is the *Non-Differential Fault Analysis*, but this is based on causing permanent damage to devices for the purpose of extracting symmetric keys (such as of DES). It must be mentioned that a trait of such attacks is that they do not require correct ciphertexts (hence, ciphertexts that were produced before the damage to the unit has occurred) [3]. This leads to the attacker being able to make use of natural faulty units (that are malfunctioning since being manufactured), without himself tampering with them.

Preventing Side Channel Attacks

This chapter presents known techniques that have been detailed in published literature. These techniques should be evaluated when defining the precautions to be taken by designers of cryptographic modules in ensuring that, to some extent, the product has the ability to resist side-channel attacks.

General Countermeasures Against All Attacks

General Data-Independent Calculations

In general, all operations that are performed by the module shall be data-independent in their time consumption. In other words, the time that operations take must be totally independent of the input data or the key data. Whenever different sub-operations are performed according to input or key bits, these sub-operations should take the same number of clock cycles.

The general feature of making the time needed for operation execution fixed for every piece of data prevents all timing attacks. This is because these attacks are

based on variations in the computation time according to input and key bits. The only input property that may have an effect on the time the operation takes is the length of the exponent in exponentiation operations. However, the length of the exponent is information with no meaningful value for the attacker, and is mostly broadly known anyway.

Blinding

Techniques used for blinding signatures can be adapted to prevent attackers from knowing the input to the modular exponentiation function [2]. This should help against any type of side-channel attack.

Even with blinding, the distribution will reveal the average time per operation, which can be used to infer the Hamming weight of the exponent. If anonymity is important or if further masking is required, a random multiple can be added to the exponent before each modular exponentiation. If this is done, care must be taken to ensure that the addition process itself does not have timing characteristics, which may reveal the random multiple. This technique may be helpful in preventing attacks that gain information leaked during the modular exponentiation operation due to electromagnetic radiation, system performance fluctuations, changes in power consumption, etc. since the exponent bits change with each operation.

A more accurate discussion of this method can be found in [2].

Avoiding Conditional Branching and Secret Intermediates

According to [4], avoiding procedures that use secret intermediates or keys for conditional branching operations will mask many SPA characteristics.

Software implementation of critical code shall not contain branching statements. Similarly, these will not contain conditional execution statements, such as IF clauses. Calculations should be performed using functions that utilize elementary operations (such as AND, OR and XOR) and not using branching and conditional execution of portions of the code.

This feature can make it extremely difficult to guess input and key values using measurements of time or power consumption. Conditional execution, which depends on input and key data, can easily reveal properties of this data if the attacker measures the time or power taken to perform certain actions. When all the lines of code are always running regardless of the input and key bits, the time and power taken to perform these actions does not depend on the data and therefore does not reveal any of its properties.

This feature prevents all types of timing attacks on asymmetric ciphers as well as some power consumption attacks.

Licensing Modified Algorithms

The most effective general solution is to design and implement cryptosystems with the assumption that information will leak. A few companies develop approaches for securing existing cryptographic algorithms (including RSA, DES, DSA, Diffie-Hellman, El Gamal, and Elliptic Curve systems) to make systems remain secure even though the underlying circuits may leak information.

Countermeasures against Timing Attacks

Adding Delays

The most obvious way to prevent timing attacks is to make all operations take exactly the same amount of time. Unfortunately this is often difficult. If a timer is used to delay returning results until a pre-specified time, factors such as the system responsiveness or power consumption may still change when the operation finishes in a way that can be detected. [2]

Also according to [2], fixed time implementations are likely to be slow; many performance optimizations cannot be used since all operations must take as long as the slowest operation.

When random delays are added, although these random delays do increase the number of ciphertexts required, attackers can compensate by collecting more measurements. The number of samples required increases roughly as the square of the timing noise [2]. So, random delays can make the attack a bit more difficult, but still possible.

Time Equalization of Multiplication and Squaring

The time taken by the unit for the performance of multiplication and for the performance of exponentiation actions should be set to be similar. Due to this quality, an attacker will not be able to learn if, when and how many multiplications are made and how many exponentiations.

The equalization can be caused by always performing both operations (multiplication and exponentiation), regardless of the operation that is required at any given time. At any stage where one of the operations is required to run, both should be executed and the aftermath of the unnecessary operation is to be silently ignored.

This technique prevents timing attacks against the exponentiation operations that are performed as a part of asymmetric encryption operations and which are subject to the most common attacks.

Countermeasures against Power Analysis Attacks

Power Consumption Balancing

Power consumption balancing techniques should be applied when possible. Dummy registers and gates should be added on which (algorithm-wise) useless operations are made to balance power consumption into a constant value. Whenever an operation is performed in hardware, a complementary operation

should be performed on a dummy element to assure that the total power consumption of the unit remains balanced according to some higher value.

Such techniques, by which the power consumption (as viewed from outside the module) is constant and independent on input and key bits, prevents all sorts of power consumption attacks such as SPA and DPA.

Reduction of Signal Size

One approach to preventing DPA attacks is by reducing signal sizes, such as by using constant execution path code, choosing operations that leak less information in their power consumption, balancing Hamming Weights and state transitions, or by physically shielding the device.

Unfortunately, such signal size reduction generally cannot reduce the signal size to zero, as an attacker with an infinite number of samples will still be able to perform DPA on the (heavily-degraded) signal. [4]

Addition of Noise

Another approach against DPA involves introducing noise into power consumption measurements. Like signal size reductions, adding noise increases the number of samples required for an attack, possibly to an unfeasibly large number. In addition, execution timing and order can be randomized to generate a similar effect [4]. Again, noise alone only increases the number of samples required, however if this increase is high enough to make the sampling unfeasible due to the number of samples required, the countermeasure works.

One suggested solution in [5] to prevent DPA attacks using noise is by adding random calculations that increase the noise level enough to make the DPA bias spikes undetectable. The results presented in [5] give some indication of how much noise needs to be added. The main goal is to add enough random noise to stop an attack, but yet to add just a minimal overhead.

Shielding

In practice, aggressive physical shielding as mentioned in [4] can make attacks unfeasible but adds significantly to a device's cost and size.

Modification of the Algorithms Design

A final approach against DPA attacks involves designing cryptosystems with realistic assumptions about the underlying hardware. Nonlinear key update procedures can be employed to ensure that power traces cannot be correlated between transactions. As a simple example, hashing a 160-bit key with SHA before using it as a key should effectively destroy partial information an attacker might have gathered about the key. Similarly, aggressive use of exponent and modulus modification processes in public key schemes can be used to prevent attackers from accumulating data across large numbers of operations.

This may solve the problem, but it does require design changes in the algorithms and protocols themselves, which are likely to make the resulting product non-compliant with standards and specifications.

COUNTERMEASURES AGAINST FAULT ATTACKS

Running the encryption twice

A possible solution (presented further in [3]) against DFA is for the unit to run the encryption twice and output the results only if these two are identical. The main problem with this approach is that it increases computation time. Also, the probability that the fault will not occur twice is not sufficiently small. Since the fault may still occur twice (especially if the fault was caused artificially), this countermeasure will only make the attack harder to implement (requiring more samples), but not impossible.

REFERENCES

1. *Introduction to Differential Power Analysis and Related Attacks* / Paul C. Kocher, Joshua Jaffe, and Benjamin Jun
2. *Timing Attacks on Implementations of DH, RSA, DSS and Other Systems* / Paul C. Kocher
3. *Differential Fault Analysis of Secret Key Cryptosystems* / Eli Biham & Adi Shamir
4. *Differential Power Analysis* / Paul Kocher, Joshua Jaffe, and Benjamin Jun
5. *Investigations of Power Analysis Attacks on Smartcards* / Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan
6. *Tamper Resistance - a Cautionary Note* / Ross Anderson & Markus Kuhn
   Informative Reference
7. *Side Channel Cryptanalysis of Product Ciphers* / John Kelsey, Bruce Schneier, David Wagner, and Chris Hall

About Discretix

Discretix is a semiconductor intellectual property company that develops and licenses advanced embedded security solutions for resource-constrained environments, such as wireless devices and smart-cards, where stringent limits apply to the cost, size and power consumption of the target devices.

Discretix technology has already been adopted by some of the major vendors of wireless baseband and application chipset, as well as smart-card IC vendors.

Discretix Technologies Ltd.

Corporate Headquarters
43 Hamelacha Street
Beit Etgarim
Poleg Industrial Zone
Netanya 42504
Israel
Tel: +972 9 885 8810
Fax: +972 9 885 8820
Email:
marketing@discretix.com

Representative in Japan:
Triangle Technologies KK
Sogo-Hirakawacho Bldg.
4F 1-4-12
Hirakawacho Chiyoda-ku
Tokyo, Japan
Te l: +81 3 5215 8760
Fax: +81 3 5215 8765
Email:
japan.info@discretix.com