Countermeasure for Differential Power Analysis using Boolean and Arithmetic masking

Wing Ng
School of Electrical Engineering and
Computer Science
Oregon State University
Corvallis, Oregon 97331
Email: ngwi@engr.orst.edu

***Abstract*— Differential Power Analysis is a side channel attack for any symmetrical cryptosystem. This paper will review the brief explanation of DPA and also some of its countermeasures. This paper will be focusing on the countermeasure using Boolean and arithmetic masking. These masking techniques will be useful for encryption schemes such as IDEA and RC6, etc. since these schemes require both Boolean and arithmetic operations.**

***Index Terms*— Differential Power Analysis, Masking Techniques, cryptanalysis**

I. Introduction

Differential Power Attack is a side channel attack by obtaining information for the secret key through the analysis the power consumption when executing the cryptographic algorithm. It was first introduced by Paul Kocher in 1998 [1]. The concept first applied to the symmetrical cryptosystems such as DES encryption algorithm and later also applied to AES algorithm. Some of the countermeasures were introduced to prevent the differential power attack including of inserting dummy code, power consumption randomization and balancing of data. However, some of these methods are proved that to be insufficient. Another countermeasure regarding using masking was introduced [2] and will be discussed in detail in this paper.

II. Differential Power Analysis

Since differential power analysis was introduced in 1998, it has been a topic that researchers are trying to overcome with. Differential power analysis based on the nature on how the transistors consume power while executing the cryptographic operations. By measuring the circuit's power consumption, attackers could find out when will the device executing the cryptographic operation such as the 16 DES rounds. From the following graph (Figure 1) from [1], attackers could trace the pattern of the power consumption curve to determine when is the device executes the DES round. By having this information, attackers could analysis this and figure out the secret key information of the DES operation.

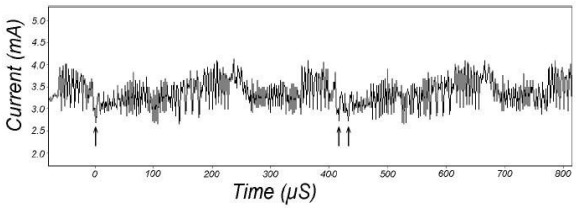

Figure 1: Power analysis trace showing DES rounds.

There are different kinds of attack regarding power analysis. The basic one referred as simple power analysis (SPA). During the SPA attack, the attacker could observe the power consumption to determine the cryptographic operations. One of the examples of SPA is to break RSA implementation by observing the differences between the multiplications and squaring operations. It also could be applied to DES to look for the differences between the permutation and shift operation.

Besides the SPA, the more powerful of the attack based on the power analysis will be the differential power analysis (DPA). The differences between SPA and DPA is that DPA is using a statistical analysis and error correction techniques to reveal the information related to the secret key rather than just by observation.

DPA attack usually applies to the portable secure device such as smart card. It is easier to collect the power consumption data from smart card since it is often to be read by the public smart card reader.

To perform DPA, it is divided into two phases: data collection and data analysis. The number of traces that needed to be measured depends on what kind of countermeasures is against to. It is possible to collect less than 15 traces to find out the DES key from the smart card. It is also possible to combine multiple samples in the same trace and analysis them to defeat some of the countermeasures. This high-order DPA is harder to block since it analysis multiple samples to reveal the information.

The following will be a belief descripion on how DPA attack applies to DES. Assume that 1000 input samples ($I_1 ... I_{1000}$) are being collected for the analysis. First, measure the power consumption curves ($C_1 ... C_{1000}$) for the first round with 1000 DES samples. Next, compute the mean curve (MC) of the power consumption for all the samples.

The next step, we need to focus on the first output bit of the S-box. Let j be the first output bit. The value of j will be

based on the 6 bits of the secret key. The attacker then makes a hypothesis on those 6 bits. Next, he or she computes the value of j from the input value (I_i) and the guessed 6 bits. All the inputs will be divided into two possible of either $j = 0$ or $j = 1$.

Next, we need to compute the MC' for $j = 0$. If MC and MC' are similar, then the guessed value is not correct and the attacker need to try another set of the 6 bits value. However, if MC and MC' have a distinct differences, then it means the guessed key is correct.

Figure 2 shows the DPA traces applies to DES operation. The first trace is the reference trace collected from the DES operation. Moreover, the other traces are produced by the guess values of the secret key. The correct guessed secret key will be one showing different from the reference trace.

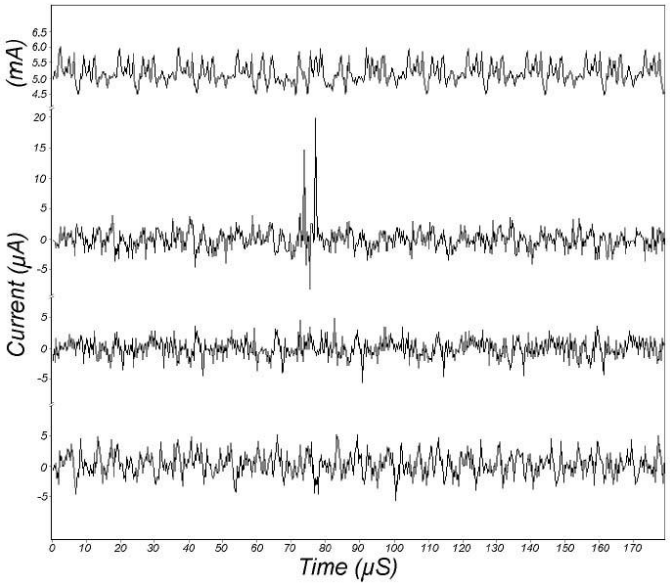

Figure 2: DPA traces for analysis DES

III. Countermeasures for DPA

Since Kocher [1] introduced DPA, it raised the attention on how to prevent the DPA. Some of the countermeasures have been proposed in order to solve this problem. Some of the countermeasures including inserting dummy code, power consumption randomization and balancing of data. Inserting dummy code could be done by insert extra codes to modify the sequence of the instructions that needed to be executed. The insertion could be added according to the parameter of the cipher in order makes it differently for each operation. Power consumption randomization could be implemented by adding extra hardware to add noise to the power consumption data. However, some of these countermeasures are not sufficient or require some trade-offs including extra hardware or more memory to execute the cryptography operations.

There are three major types of counter measures: (1) random timing shifts, (2) replacing some of the critical instructions for the cryptographic algorithm, and (3) using an algorithm to encode the original data. Most of the countermeasures are related to encoding the power consumption information. Boolean masking and arithmetic masking is one of the countermeasures that have been proposed under the encoding power consumption data category.

IV. Boolean and Arithmetic masking

The masking techniques only apply to the operation where it could produce a masked output based on the masked input. There are two masking techniques: Boolean masking and Arithmetic masking. Boolean masking involves randomizing the Boolean operation using some random variables and arithmetic masking is similar to that.

Boolean masking: $x' = x \oplus r$
Arithmetic masking: $x' = (x - r) \pmod{2^k}$

x' referred as the masked output and r represented the random variable.

Since some of the cryptographic algorithms such as IDEA, RC6 and TWOFISH, etc. involves both Boolean and arithmetic operations, an effective method would be needed to switch between Boolean masking to arithmetic masking and via verse. Both Goubin[3] and Coron and Tchulkine[4] proposed similar idea for a more efficient way for Boolean and arithmetic switching.

By applying the masking technique, the original data will be split into multiple shares and the masked data will be more evenly distributed than the original. Due to this reason, attackers need to evaluate multiple distributions for each shares and the computation time will be increased exponentially. However, applying masking technique also take more memory and computation time to mask and unmask the data. Sometimes, it also involves the conversion between different kinds of masking. Switching Boolean and arithmetic masking could be a problem.

V. Definition of Boolean and arithmetic masking

To perform a Boolean masking, we need to mask the data as $x' = x \oplus r$ where r is the random variable and it is uniformly distributed. To describe this in more detail, we can see if $x_3 = x_1 \oplus x_2$, then the masked value x'_1 and x'_2 are referred as $x'_1 = x_1 \oplus r_1$ and $x'_2 = x_2 \oplus r_2$. Moreover, $x'_3 = x'_1 \oplus x'_2$ where $r_3 = r_1 \oplus r_2$ and then $x_3 = x'_3 \oplus r_3$.

Arithmetic masking is doing as the similar way. $x_3 = x_1 + x_2$ and the mask value for x'_1 and x'_2 will be $x_1 + r_1$ and $x_2 + r_2$. Therefore, when computing the masked value for x'_3, we have $x'_3 = x'_1 + x'_2$ and $r_3 = r_1 + r_2$. As a result, $x_3 = x'_3 + r_3$.

VI. Switching between Boolean masking and arithmetic masking

In [3] Goubin already proposed an effective switching method between the Boolean masking to arithmetic masking. However, the switching from arithmetic masking to Boolean masking is not that efficient. Therefore, in [4], Coron and Tchulkine modified the algorithm in order to reduce the execution time. We will discuss the following based on the algorithm that was proposed in [4]. Both of the methods are similar except in [4] proposed to use a pre-computed lookup table.

VII. Boolean to Arithmetic masking

In [3] had proposed an efficient algorithm for conversion from Boolean masking to Arithmetic. The algorithm will be described as follow:

Input: (x', r) as $x = x' \oplus r$
Output: (A, r) as $x = A + r$

x' is the masked value from Boolean masking and A is the masked value from arithmetic masking. This algorithm requires two more variables besides the random variable r.

Generate the random value for both r and R

$T \Leftarrow x' \oplus R$
$T \Leftarrow T - R$
$T \Leftarrow T \oplus x'$
$R \Leftarrow R \oplus r$
$A \Leftarrow x' \oplus R$
$A \Leftarrow A - R$
$A \Leftarrow A \oplus T$

Goubin's algorithm for Boolean to Arithmetic masking

This conversion only requires seven operations including 5 XOR and 2 subtractions. However, the algorithm for arithmetic masking to Boolean masking takes more work.

Input: (A, r) as $x = A + r$
Output: (x', r) as $x = x' \oplus r$

x' is the masked value from Boolean masking and A is the masked value from arithmetic masking. This algorithm requires two more variables besides the random variable r.

Generate the random value for both r and R

$T \Leftarrow 2T$
$x' \Leftarrow R \oplus r$
$C \Leftarrow R \wedge x'$
$x' \Leftarrow T \oplus A$
$R \Leftarrow R \oplus x'$
$R \Leftarrow R \wedge r$
$C \Leftarrow C \oplus R$
$R \Leftarrow T \wedge A$
$C \Leftarrow C \oplus R$
for $k = 1$ to $K - 1$ do $R \Leftarrow T \wedge r$
$R \Leftarrow R \oplus C$
$T \Leftarrow T \wedge A$
$R \Leftarrow R \oplus T$
$T \Leftarrow 2R$
end for $x' \Leftarrow x' \oplus T$

Goubin's algorithm for Arithmetic to Boolean masking

This algorithm needs $(5K + 5)$ elementary operations and it is not as easy as the algorithm for Boolean to arithmetic masking.

VIII. Proposed algorithm for Arithmetic masking to Boolean masking

In the proposed algorithm from Coron and Tchulkine for the switching between Boolean masking and arithmetic masking, it requires to pre-compute a table for all the value for the switching between Boolean and arithmetic masking. Since the output value computed already, when executing the actual switching will be a lot easier since it only need to pick the value from the table instead of computing it each time. However, this idea only will be benefit if the pre-computed table would be require to calculate once for the whole cryptographic operation. Otherwise, this algorithm will slower down the execution time since it takes time to create the lookup table.

For the proposed algorithm, it requires four basic steps. First, generating the Boolean masked value from arithmetic masking into a size of 2^k bits table where k is the number of the bits of the input.

Second, performing the conversion from arithmetic masking to Boolean masking based on the lookup table. Since it is required to generate a lookup table, this method is more suitable for small k.

For a larger bits input, the size of k will be split into smaller size of two or more shares and then combine them together after the conversion. However, it is required to compute an extra lookup table for the carry when combining the two shares together. Another lookup table for the size of 2^k is generated. The details of the larger value will be shown as follow:

First, we split the inputs into $l * k$ bits. There are two input variables in arithmetic masking A and R from $x = A + R \pmod{2^{l*k}}$. We need to covert them into the Boolean masking form such as $x = x' \oplus R$. Split R into $R_1 \| R_2$ and R_1 with the size of $(l-1) * k$ bits and R_2 of size k bits. Generate a random k bits r, we get

$$A \Leftarrow (A - r) + R_2 \pmod{2^{lk}}$$

We also need to split A into $A_1 \| A_2$ as A_1 has the same size of R_1 and A_2 with size of k. Then, we compute as follow

$$x = (A_1 \| A_2) + (R_1 \| r) \pmod{2^{lk}}$$

Since the variable is split into two shares, if $A_2 + r \geq 2^k$, then $A_1 \Leftarrow A_1 + 1 \pmod{2^{(l-1)*k}}$. This computation acts as the carry from the addition of $A_2 + r$ and added to A_1.

Next, we need to split x into $x_1 \| x_2$ and x_1 has the size same as A_1 and R_1. Then, we could compute the followings:

$$x_1 = A_1 + R_1 \pmod{2^{(l-1)*k}} \text{ and } x_2 = A_2 + r \pmod{2^k}$$

After we got these equations, we could use the pre-computed lookup table to find out the Boolean masked value such that $x'_2 \leftarrow G[A_2]$ where G is the pre-computed lookup table and A_2 is the index of G. For x'_1, we could use a recursive method to find out its value. After obtaining the values for x'_1 and x'_2, we are able to get x' such as $x' = x'_1 \| x'_2$. The output of the conversion will be $x = x' \oplus R$.

Coron and Tchulkine also compare its efficiency along with Goubin's algorithm (see Table 1) for different input bits in order to show the difference on how their algorithm will be benefit when performing on a small input size.

	$T_{8,8}$	$T_{8,32}$	$T_{32,8}$	$T_{32,32}$	$G_{8,8}$	$G_{8,32}$	$G_{32,8}$	$G_{32,32}$
Pre-computation time	64	64	64	64	0	0	0	0
Conversion time	10	10	76	40	45	45	660	165
Table size	32	32	32	32	0	0	0	0

Table 1: Comparison with Goubin's algorithm

From the given Table 1, $T_{i,j}$ referred as Coron and Tchulkine's algorithm and $G_{i,j}$ referred as Goubin's algorithm. The variable i represents the number of bit of the input and j represents the number of bit of the system. According to Coron and Tchulkine's comparison, their algorithm performs better for the smaller system such as an 8 bit microprocessor. It is useful for the smart card since the 8 bit microprocessor is a common platform for the smart card. However, when it moves to a larger system with a larger input bit, Goubin's algorithm will be a better choice. This difference is closely related to the size of the pre-computed lookup table.

IX. Conclusion

Boolean and arithmetic masking sounds like a possible way to prevent DPA, however, the masking technique also involves some overhead including taking more memory and more computation time for conversion when performing the cryptographic algorithm. However, these overhead sometime are necessary in order to prevent the side channel attack to ensure security. Both of the hardware and software solution should be considered to achieve better efficiency along with all the trade-offs.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," pp. 388–397, 1999.

[2] J. Coron and L. Goubin, "On boolean and arithmetic masking against differential power analysis," *Proceedings of CHES 2000*, vol. 1965, pp. 231–237, 2000.

[3] L. Goubin, "A sound method for switching between boolean and arithmetic masking," *Proceedings of CHES 2001*, vol. 2162, pp. 3–15, 2001.

[4] J. Coron and A. Tchulkine, "A new algorithm for switching from arithmetic to boolean masking," *Proceedings of CHES 2003*, vol. 2779, pp. 89–97, 2003.