Sign-Based Differential Power Analysis

Roman Novak

Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia,
Roman.Novak@ijs.si

Abstract. [1]Differential Power Analysis (DPA) by Paul Kocher et al. is expanded with the information that is hidden in the sign of power biases. The latter reveal values that collide with the DPA target value within the circuitry. With the help of cross-iteration comparisons, the interpretation of those values can provide significant amounts of the information required to reverse engineer secret algorithm. We have successfully launched a demonstration attack on a secret authentication and session-key generation algorithm implemented on SIM cards in GSM networks. The findings provide guidance for designing tamper resistant devices that are secure against this kind of attack.

1 Introduction

Any real cryptographic device provides more information to a determined adversary than just the input plaintext and output ciphertext. This side-channel information is available as the timing of operations [1], power consumption of the devices [2], electromagnetic emanations [3], etc. Very little side-channel information is required to break many common ciphers. Non-invasive attacks and their accompanying countermeasures have been studied extensively over the past few years. Systems that rely on smartcards are of particular concern.

Most of the side-channel based methods deal with the extraction of cryptographic material, such as keys, from the implementations of well-known algorithms. A fundamental rule of cryptography is that one must assume that the cryptanalyst knows the general method of encryption used. Experts have learned over the years that the only way to ensure security is to follow an open design process, encouraging public review to identify flaws while they can still be fixed. However, many cryptographic algorithms are still kept secret. For instance, GSM network operators use an updated version of COMP128-1, designated as COMP128-2, as an authentication algorithm, but the algorithm remains unpublished. Some network operators even develop a proprietary algorithm in secrecy. In either case, the algorithm used has not been publicly reviewed.

The purpose of this paper is to show that secret algorithms offer very little protection. We expand the well-known Differential Power Analysis (DPA) attack [2] with the information that is hidden in the sign of power biases. An adversary

[1] K. Chae, M. Yung (Eds.): WISA 2003, LNCS 2908, pp. 203-216, 2003.
©Springer-Verlag Berlin Heidelberg 2004

can use a method similar to the proposed Sign-based Differential Power Analysis (SDPA) to reverse engineer secret algorithms. A sign of power biases has been already used in other contexts, e.g. in a power based attack on the Twofish reference code [4], but its use for reverse engineering is new. Experiments on reverse engineering using side-channel information have been conducted by other authors, i.e. in [5] neural networks are used to automate reverse engineering of software code. Strength comparison of known methods has not been performed yet.

An example is given on the secret authentication and session-key generation algorithm implemented on SIM (Subscriber Identity Module) cards that are used in GSM networks. Specification of the algorithm was not available. Correctness of the restored algorithm was checked by means of plaintext/ciphertext pairs. As the algorithm is not well-known COMP128-1, only small parts of the code are given in order to keep the algorithm secret and be clear enough.

The rest of the paper is structured as follows. Section 2 gives a short introduction to power analysis techniques. In Sect. 3 we introduce the theoretical basis of Messerges et al. for the DPA attack [6], invented by Kocher et al. [2]. We expand the method to create a more powerful SDPA attack in Sect. 4. In Sect. 5 the leaked information that can be captured by the SDPA is detailed. In order to improve the interpretation of SDPA vectors, cross-iteration comparisons are suggested in Sect. 6. An example is given on an unknown GSM authentication algorithm. Usually, supplementary methods have to be employed to completely restore the algorithm under attack. Some of them are mentioned in Sect. 7. Section 8 expands the SDPA with iteration independent SDPA matrices and shows how they can be used to extract permutations from side-channel information. Countermeasures against the proposed side-channel attack are discussed in Sect. 9. The findings are summarised in Sect. 10.

2 Power Analysis

Smart cards consist of logic gates, which are basically interconnected transistors. During operation, charges are applied to or removed from transistor gates. The sum of all charges can be measured through power consumption, on which power analysis techniques are based.

Several variations of power analysis have been developed [2, 7]. The power consumption measurements of smart card operations are interpreted directly in Simple Power Analysis (SPA). SPA can reveal hidden data in algorithms in which the execution path depends on the data being processed. More advanced techniques, like Differential Power Analysis (DPA) and Inferential Power Analysis (IPA), allow observation of the effects correlated to the data values being manipulated.

Power analysis attacks have been known for a while and effective countermeasures exist that pose difficulties, even to a well-funded and knowledgeable adversary [8, 9, 6]. However, many providers for cryptographic tokens are still convinced that implementation of secret algorithms provide a sufficient level

of protection against low-cost side-channel attacks. The results presented here speak against attempts to establish secrecy by keeping cryptographic algorithms undisclosed.

3 DPA Attack

A DPA attack begins by running the encryption algorithm for N random values of plaintext input. For each of the N plaintext inputs, PTI_i, a discrete time power signal, $P_i[j]$, is collected. The corresponding ciphertext output may also be collected. The power signal $P_i[j]$ is a sampled version of the power being consumed during the portion of the algorithm that is under attack. The index i corresponds to the PTI_i that produced the signal and the index j corresponds to the time of the sample. The $P_i[j]$ are split into two sets using a partitioning function, $D(\cdot)$:

$$\begin{aligned} P_0 &= \{P_i[j] | D(\cdot) = 0\} \\ P_1 &= \{P_i[j] | D(\cdot) = 1\} \ . \end{aligned} \tag{1}$$

The partitioning function can take plaintext input or ciphertext output as a parameter. The next step is to compute the average power signal for each set:

$$\begin{aligned} A_0[j] &= \tfrac{1}{|P_0|} \textstyle\sum_{P_i[j] \in P_0} P_i[j] \\ A_1[j] &= \tfrac{1}{|P_1|} \textstyle\sum_{P_i[j] \in P_1} P_i[j] \ , \end{aligned} \tag{2}$$

where $|P_0| + |P_1| = N$. Note that it is not necessary for the sets P_0 and P_1 to be exactly equal. By subtracting the two averages, a discrete time DPA bias signal, $T[j]$, is obtained:

$$T[j] = A_0[j] - A_1[j] \ . \tag{3}$$

Selecting an appropriate function $D(\cdot)$ results in a DPA bias signal that can be used to verify guessed portions of the secret key. For instance, the partitioning function can differentiate power traces into two groups based on the predicted value of a bit of some intermediate variable, which in turn depends on a portion of the secret key. At some point during execution, the software needs to compute the value of this bit. When this occurs, or when any data containing this bit is manipulated, there will be a slight difference in the amount of power dissipated, depending on whether this bit is a 0 or a 1. As the number N of PTI inputs is increased, $T[j]$ converges to the expectation equation:

$$\lim_{N \to \infty} T[j] = E\{P_i[j] | D(\cdot) = 0\} - E\{P_i[j] | D(\cdot) = 1\} \ . \tag{4}$$

If enough PTI samples are used and a correct guess as to the value of the intermediate variable has been made, $T[j]$ will show power biases of ε at times j' and will converge to 0 all other times. Time indices j' are those indices at which the instructions manipulating the observed bit occur. Due to weakly correlated events, $T[j]$ will not always converge to 0 for $j \neq j'$; however, the largest biases will occur at times j'. Figure 1 shows power biases for a correct guess on the value of the secret and a signal without biases for an incorrect guess.

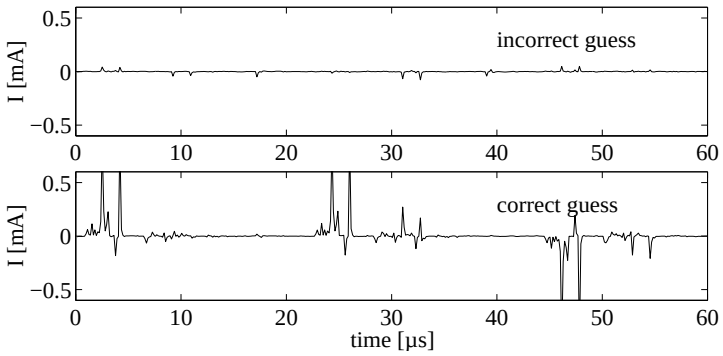

Fig. 1. Example of experimental DPA power biases

4 Sign of Power Biases

The DPA method as described does not put any significance on the sign of biases in $T[j]$. However, the sign carries additional information about the algorithm under attack. Biases are of different signs, positive or negative, as can be clearly seen from Fig. 1.

The different sign of biases can be explained by a closer look at semiconductor technology, the source of power dissipation. In [6] a simple model of CMOS circuit is proposed as a basis for understanding power dissipation measurements. CMOS logic has almost no static power dissipation, but there is dynamic power dissipation. The gate of a transistor forms a small capacitor. When switching an input, not only the transistor gate capacities but also the whole wire needs to be charged or discharged. Note that the wires connecting the gates have even bigger capacities than the gates themselves. There are also short-circuit currents during switching due to the way CMOS logic is designed.

Suppose the partitioning function $D(\cdot)$ returns the value of a bit that occurs on the input of CMOS gate at time j'. The impact on power dissipation will occur only if the state of the input changes. Let the initial state of the input be 0. There is then an increase in power dissipation in power traces selected to set P_1, hence negative power bias at time j'. On the other hand, when the initial state of the CMOS input is 1, the reverse is true and positive power bias is observed at time j'. The above explanation holds when the difference of the two sets is computed as in (3). If the difference is computed in a different way, the signs have to be interpreted accordingly.

It is evident from the above that the sign of DPA bias carries information about the initial state of the circuitry. Furthermore, a similar effect on power dissipation may be observed when the circuitry that already holds the bit under observation is assigned the next value. Again, the sign of DPA bias reveals this next value.

Suppose the correct guess as to the value of the intermediate has been made, or that the intermediate is not secret at all. Note that the term 'intermediate'

refers to the value of the intermediate variable within the algorithm. By performing sign analysis of power biases for all bits of the intermediate, values can be revealed that collide with the observed value in the same circuitry.

Definition 1 (of the SDPA vector). *Let $D_{I_b}(\cdot)$ be a partitioning function that splits power traces as regard to the value of the b-th bit of an intermediate I and the j' time index at which DPA power bias is observed for at least one bit of the intermediate. Let $T_{I_b}[j']$ be a DPA bias signal for $D_{I_b}(\cdot)$. An SDPA vector $\mathbf{s}_{j'} \in \{0, 1, \sim\}^{n+1}$ is defined as*

$$\mathbf{s}_{j'} = [sgn(T_{I_n}[j']), \ldots, sgn(T_{I_1}[j']), sgn(T_{I_0}[j'])] \ , \tag{5}$$

where the bit length of the intermediate I is $n+1$ and sgn is defined as

$$sgn(x) = \begin{cases} 1 & x > threshold \\ 0 & x < -threshold \\ \sim & otherwise \ . \end{cases} \tag{6}$$

Definition 2 (of the SDPA value). *When $\mathbf{s}_{j'}$ is completely defined, i.e. $\mathbf{s}_{j'} \in \{0, 1\}^{n+1}$, an SDPA value can be extracted by computing the dot product*

$$s_{j'} = \mathbf{s}_{j'} \cdot [2^n, ..., 2, 1] \ . \tag{7}$$

Note that the intermediate can be handled internally as a bit complemented value, in which case the SDPA value is also bit complemented.

5 Leaked Information

Basically DPA is used as a tool for testing various hypotheses about secret values in cryptographic algorithms. This is done through selection of the partitioning function. On the other hand, SDPA is performed on known variables within an algorithm in order to deduce further knowledge about the algorithm. In this way secret values can be detected as well, however, they are revealed through collision with known values.

Empirical evidence is provided on the kind of information that can be revealed through SDPA in the following sections. The method has been validated on an unknown authentication and session-key generation algorithm (A3A8) implemented on SIM card. The algorithm's specification was not made available in advance. All that was known was that the algorithm was not COMP128-1. The experiment was part of a larger ongoing project in which the significance of the side-channel information is evaluated in various reverse engineering techniques.

Experiments in [6] showed that, in a typical smart card, a large portion of the power dissipation occurs in the gates attached to internal busses. However, when a precharged bus design is used, SDPA values caused by bus activity are of limited usability because the bus is driven to a constant value before and after each transfer. This is not the case for internal busses and other circuitry, including microcontroller registers. We could not establish the exact source of SDPA

information as we were experimenting with smart card of unknown architecture. Future research into this area is required.

When the algorithm is secret, the plaintext input can be selected as a known intermediate variable. The input can usually be split into n-bit blocks, where n depends on the microcontroller architecture; today 8-bit microcontroller architecture is the most common. Applying SDPA to input blocks results in a set of SDPA vectors $\mathbf{s}_{j'}$ with different time indices j'. Not all of them are completely defined.

It is difficult to deduce the meaning of SDPA values, as there can be many explanations. For instance, a constant may be a real constant used by the algorithm, but it can also be the memory address of a variable, value of a variable, opcode, or just the effect of a precharged bus. Other more complex explanations are possible. Note that events that correlate with the observed value, i.e. selected bit of the intermediate, also result in non-zero power biases.

Moreover, the collision of the observed value with its transformed counterpart produce very distinct SDPA vectors, which can be considered as a signature of the transformation. First results based on simulation suggest that various arithmetic and logic operations may be identified by analysing SDPA vectors. For example, suppose that input value i collides with its shifted counterpart, $i \gg 3$. Then the following SDPA vector $\mathbf{s}_{j'}$

$$\mathbf{s}_{j'} = [0, 0, 0, \sim, \sim, \sim, \sim, \sim] \tag{8}$$

is expected, where the number of consecutive zeros equals the number of places shifted to the right.

Proof. The upper three bits of input collide with zero due to a right shift, therefore, zero components are expected. On the other hand, the lower bits of input collide with shifted bits. There is a transition on a gate if one of the collided bits is 1 and another is 0. When input values are evenly distributed, approximately half the power traces in each partition show power bias due to transition. Therefore, the DPA power signal for the lower five bits at time j' converges to zero, hence neither positive nor negative bias. □

Note that the same signature would be obtained if zero collided with masked input. Signatures for other typical logical and arithmetic operations may be derived. They are very dependent on the value with which the transformed value collides. Similar signatures may result from different transformations, while some transformations cannot be identified in this way. For instance, when the rotated value collides with its original, the DPA power bias limits to zero. Furthermore, many signatures, like those for addition and multiplication, have an effect on the size of a bias and not only on its sign. When the transition of some bits leak more information than others, i.e. power bias is dependent on bit position [10], the interpretation of magnitudes in addition to signs may appear difficult. Further research is required in order to include the information about magnitudes into signatures.

6 Cross-Iteration Analysis

Many ciphers implement complex compositions of simple operations through different levels of iterated code. In order to correctly interpret fully defined SDPA vectors, the investigation of similar vectors from multiple iterations can be very helpful. We define the SDPA matrix that captures sign information for cross-iteration comparisons.

Definition 3 (of the SDPA matrix). *Let m be the number of iterations in which similar patterns of SDPA responses have been observed. Suppose the iterations are of fixed length l. An SDPA matrix $\mathbf{S}_{j'}$ is a collection of transposed SDPA vectors*

$$\mathbf{S}_{j'} = [\mathbf{s}^T_{j'}, \mathbf{s}^T_{j'+l}, \mathbf{s}^T_{j'+2l}, \ldots, \mathbf{s}^T_{j'+(m-1)l}] \ , \tag{9}$$

where time index j' refers to the first iteration. The SDPA vectors are computed for intermediates that are expected to be processed within each iteration.

We have applied SDPA to an unknown GSM authentication algorithm that has been deployed by different service providers. The algorithm is a keyed hash function. It takes a 16-byte key (128 bits) and 16-byte of data (128 bits) to output a 12-byte (96 bits) hash. The key $k_0 - k_{15}$, as used in the GSM protocol, is unique to each subscriber and is stored in the SIM card. The input data $i_0 - i_{15}$ is a random challenge supplied by the base station. The first 32 bits of the hash are used as a response to the challenge and sent back to the base station. The remaining 64 bits are used as a session key for voice encryption using the A5 algorithm.

The following SDPA matrices are identified at the beginning of the algorithm:

$$\begin{aligned}
\mathbf{C}_0 &= [0,0,0,0,\ldots,0,0,0,0] \\
\mathbf{CNT}_{0-15} &= [0,1,2,3,\ldots,12,13,14,15] \\
\mathbf{KEY} &= [76,157,145,129,\ldots,217,109,31,224] \\
\mathbf{CNT}_{0-15} \oplus \mathbf{KEY} &= [76,156,147,130,\ldots,213,96,17,239] \\
\mathbf{CNT}_{69-84} \oplus \mathbf{KEY} &= [9,219,214,201,\ldots,136,63,76,180] \ ,
\end{aligned} \tag{10}$$

where the matrix columns are written as the SDPA values. 8-bit blocks of plaintext input $i_0 - i_{15}$ were selected as known intermediate variables, one per iteration. The time index j' of the above matrices is omitted as the matrices repeat several times. In Fig. 2 a temporal ordering of the matrices within the algorithm's first iterations is shown.

When only one iteration is observed, the SDPA values are just different constants that collided with the input. The matrix representation gives more information. For instance, $\mathbf{CNT}_{0-15}$ can be interpreted as a loop counter in collision with the plaintext input. This explanation is very likely as both input and counter are handled within a typical loop. In the middle of the iteration a SDPA matrix that appears to contain random SDPA values is identified three times. Our prediction that a key $k_0 - k_{15}$ collided with the input has proven correct, hence the label **KEY**. At the end of the iteration two different matrices were

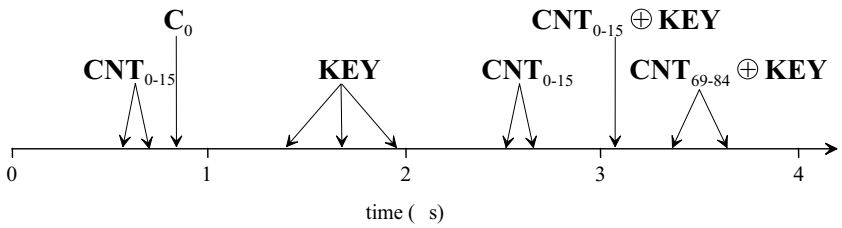

Fig. 2. Temporal ordering of the SDPA matrices within the first iterations

identified. The first can be calculated as a bitwise exclusive or between $\mathbf{CNT}_{0-15}$ and **KEY**, $\mathbf{CNT}_{0-15} \oplus \mathbf{KEY}$, and the second as $\mathbf{CNT}_{69-84} \oplus \mathbf{KEY}$. It is highly unlikely that these values actually collided with the input; however, the same matrices are obtained if $\mathbf{KEY} \oplus i_0 - i_{15}$ collides with $\mathbf{CNT}_{0-15}$ and $\mathbf{CNT}_{69-84}$, respectively. This is due to the property of exclusive or. When SDPA is performed on an intermediate xored by a constant C without modification of the partitioning function, the partitions P_0 and P_1, and thus the signs, are swapped for all ones in C; the same effect would have the exclusive or between the initial state and C. There is also a third option according to which $\mathbf{CNT}_{0-15} \oplus i_0 - i_{15}$ collided with **KEY**, but it is less probable. The $\mathbf{CNT}_{69-84}$ may show the memory range where the results of the computation are stored. Other explanations are also possible. From the above conclusions, one can write the Alg. 1 that would cause a similar response to SDPA.

Algorithm 1. Initial computation in unknown algorithm
FOR j from 0 to 15
    $x[j] = i[j] \oplus k[j]$
END FOR

Note that the flow of the operations is in agreement with the temporal ordering of the SDPA matrices. First, the input is read. After that, the key is accessed. Computation of the exclusive or follows. Finally, the results are stored in the memory.

In the above example, SDPA reveals information about keys, counters, memory ranges, operations and their temporal ordering.

7 Supplementary Methods

As shown by a simple example, SDPA can be used in the reverse engineering attempts of an adversary to reveal secret code. After new intermediate values have been discovered, SDPA is performed on those values instead of on blocks of plaintext input. The intermediates can be deduced from the information in SDPA matrices and by testing the most probable hypotheses about transformations. SDPA signatures of various logical and arithmetic operations can provide significant help in selecting the right hypothesis.

Supplementary methods have to be employed to completely restore the algorithm under attack. For instance, simple power analysis (SPA) and the use of correlation techniques may help in identifying the algorithm's loops.

The next major difficulty in restoration attempts are substitution blocks, which are usually implemented as lookup tables. Modern cryptography uses substitution as a building block in complex compositions of strong ciphers. Substitution tables are considered to provide a high security level because they contribute effectively to data diffusion. A substitution block is a very effective countermeasure against SDPA as it prevents intermediates from being tracked through the algorithm. However, many implementations of lookup operation are insecure. An attack on substitution blocks is proposed in [12]. The attack is based on identifying equal intermediate results from power measurements while the actual values of these intermediates remain unknown.

We managed to completely restore the unknown GSM authentication algorithm using the above methods. In Fig. 3 a detail of the algorithm is shown that includes four lookup operations. Only a small fraction of the sixteen similar iterations is shown. The code is executed at the beginning of the algorithm and can be restored only by combining SDPA, SPA and substitution block attack.

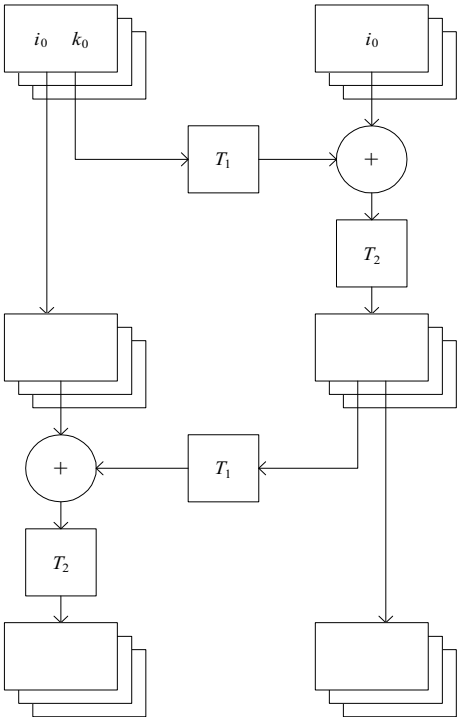

Fig. 3. Computation in the GSM authentication algorithm

8 Iteration Independent SDPA

The code from Fig. 3 is repeated five times within eight iterative loops. Each inner iteration is followed by a permutation while each outer iteration completes with a compression. The compression is a permutation on a subset of bits.

We define iteration independent SDPA matrices and show how they can be used to extract permutation or compression from side-channel information. We demonstrate the method on a detail of the compression from Fig. 4.

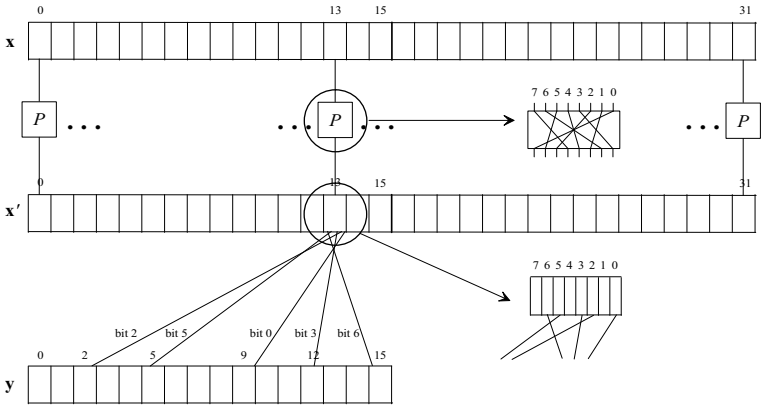

Fig. 4. Detail of the outer compression in the GSM authentication algorithm

The algorithm was extracted from the outer loop of our GSM authentication algorithm. A 32-byte (256 bits) vector $\mathbf{x}$ is compressed in a 16-byte (128 bits) vector $\mathbf{y}$. First, bit permutations within the components of $\mathbf{x}$ are performed using the permutation P. Half the bits are then extracted from the vector $\mathbf{x}'$ and recomposed in $\mathbf{y}$. In the last step, the position of each bit within the components of $\mathbf{y}$ remains the same as within the components of $\mathbf{x}'$.

Definition 4 (of the iteration independent SDPA matrix). *When specific intermediates to be processed within each iteration cannot be determined, SDPA vectors for a fixed intermediate could be computed regardless of the iteration. Let $\mathbf{S}^I_{j'}$ designate an iteration independent SDPA matrix computed on fixed intermediate I.*

Iteration independent matrices have usually only one column defined because the intermediate is often processed only in one, previously unknown iteration. This type of matrix can be used to reveal permutations from the side-channel information. Again, sign information can be an additional help in making hypotheses about the algorithm.

The following matrices were obtained when iteration independent SDPA was applied to x_{13}:

$$
\begin{aligned}
\mathbf{A}_1^{x[13]} &= [\sim, \ldots, \sim, 13, \sim, \sim, \sim, \sim, \sim, \sim] \\
\mathbf{B}_1^{x[13]} &= [\sim, \ldots, \sim, 0, \sim, \sim, \sim, \sim, \sim, \sim] \\
\mathbf{C}_1^{x[13]} &= [\sim, \ldots, \sim, \mathbf{c}_1^T, \sim, \sim, \sim, \sim, \sim, \sim] & \mathbf{c}_1 &= [0,0,0,0,\sim,0,0,0] \\
\mathbf{D}_1^{x[13]} &= [\sim, \ldots, \sim, \mathbf{d}_1^T, \sim, \sim, \sim, \sim, \sim, \sim] & \mathbf{d}_1 &= [\sim,\sim,\sim,\sim,0,\sim,\sim,\sim] \\
\mathbf{A}_2^{x[13]} &= [\sim, \sim, 13, \sim, \ldots, \sim] \\
\mathbf{B}_2^{x[13]} &= [\sim, \sim, 0, \sim, \ldots, \sim] \\
\mathbf{C}_2^{x[13]} &= [\sim, \sim, \mathbf{c}_2^T, \sim, \ldots, \sim] & \mathbf{c}_2 &= [0,0,0,0,0,0,\sim,0] \\
\mathbf{D}_2^{x[13]} &= [\sim, \sim, \mathbf{d}_2^T, \sim, \ldots, \sim] & \mathbf{d}_2 &= [\sim,\sim,\sim,\sim,\sim,\sim,0,\sim] \ .
\end{aligned}
\tag{11}
$$

Note that the vector $\mathbf{x}$ is already known to an adversary. Temporal ordering of the matrices is given in Fig. 5. Only a subset of matrices is shown in order to demonstrate the reconstruction of the compression algorithm. Below the time axis the positions of other matrices are marked for the intermediates $x_0 - x_{31}$. Eight groups were detected, but only three are shown.

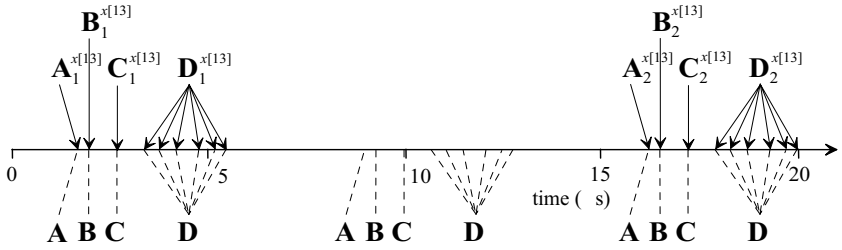

Fig. 5. Temporal ordering of the iteration independent matrices

From the above, several conclusions about the algorithm can be made. Here, we give the most probable explanation, which led to the reconstruction of the compression algorithm in Fig. 4.

The compression is performed in a part of the algorithm where 16 iterations were detected; hence 16 columns in the matrices. The intermediate, in our case x_{13}, appears in different iterations, i.e. different columns are defined. We concluded that a sort of permutation is performed, with each iteration computing one component of the output vector $\mathbf{y}$. Eight groups of matrices were identified within the iterations; only some groups were present for a given intermediate. We made an assumption that the 8-bit components of $\mathbf{y}$ are computed one bit per group.

Each group starts with matrix $\mathbf{A}$ for which the only defined vector contains the address of the intermediate. Confirmation that the value is really an address, and not just a constant, could be found in the matrices for other intermediates. The collision of the intermediate with its address suggests that, within each group, an independent computation is performed on the intermediate value.

Matrices $\mathbf{C}$ and $\mathbf{D}$ can be signatures of various masking operations. For instance, when a constant collides with an intermediate that has been previously masked, only corresponding components of the SDPA vector are defined. Therefore, bit 3 of x_{13} could be extracted in the 10th iteration, as one can deduce from the matrices $\mathbf{C}_1^{x[13]}$ and $\mathbf{D}_1^{x[13]}$. A similar conclusion, based on the matrices $\mathbf{C}_2^{x[13]}$ and $\mathbf{D}_2^{x[13]}$, can be made for bit 1 in the 3rd iteration.

The temporal ordering of the extracted bits within the iteration can be the ordering of bits in the result, i.e. iteration j computes value y_j by bit composition from bit 0 to bit 7. Consequently, bit 3 of x_{13} maps to bit 0 of y_9 while bit 1 of x_{13} maps to bit 2 of y_2.

The described response to iteration independent SDPA could be achieved by implementation of Alg. 2.

Algorithm 2. Restored compression algorithm
FOR j from 0 to 15
$y[j] = (x[b_0[j]] \;\&\; 0x08) \gg 3) \;|$
$(x[b_1[j]] \;\&\; 0x40) \gg 5) \;|$
$(x[b_2[j]] \;\&\; 0x02) \ll 1) \;|$
$\ldots$
END FOR

The first level permutation is performed by masking and shifting operations. $b_0 - b_7$ are the second level permutation tables and can be defined from the information in the iteration independent SDPA matrices.

9 Countermeasures

Designers must not rely on secrecy of the algorithm because the algorithm may be reverse-engineered in the presence of side-channel information leakage. Only low-cost equipment is needed to perform this kind of attack. Even when well-tested public algorithms are used, there is a chance that SDPA may reveal sensitive information or break a layer of protection. Techniques for preventing SDPA attacks, as described in this paper, fall roughly into two categories.

The first approach is to prevent information leakage, using the general techniques that protect the algorithm as a whole. Well-known techniques are signal size reduction, introducing noise into power consumption, randomised execution timing and order, balancing state transitions, key use counters, physically shielding the device, blinding, multithreaded applications, etc [13, 2, 8, 9].

On the other hand, many techniques can be used to bypass or compensate for these countermeasures. We suggest the use of as many redundant countermeasures as available resources permit, because many of the countermeasures can be compensated for if they are implemented alone [11]. The best way to eliminate an attack would be adherence to the cardinal principle [9], which is hard to achieve. According to the cardinal principle, if differential attacks are to be completely eliminated, relevant bits of all intermediate cycles and their

values should be statistically independent of the inputs, outputs and sensitive information.

A second category of countermeasures against SDPA attacks involves measures that prevent leakage of sensitive information through the sign of power biases. Due to the presence of unexpected sign-based leakages, a comprehensive vulnerability assessment has to be an integral part of any effort to protect specific implementation. In order to prevent collisions of sensitive information with well-known intermediates, the use of precharged busses is suggested. A similar action should be carried out in all parts of the circuitry where sign-based leakage is observed. An even better solution is to feed registers and busses with random values. The countermeasures should be implemented in hardware, however software based solutions can also provide a significant level of protection.

10 Conclusion

The realities of a physical implementation can be extremely difficult to control. The level of tamper resistance offered by any particular product can be measured by the time and cost penalty that the protective mechanisms impose on the attacker. When the method is known, a side-channel attack on a moderately protected smartcard typically requires a few minutes to several hours to complete, while the cost of the sampling equipment falls in the range from several hundred to several thousand dollars.

We have introduced a sign extension to the differential power analysis. The sign of DPA biases reveal values that collide with the DPA target value within the circuitry. Basically, DPA is used as a tool for testing various hypotheses about secret values in cryptographic algorithms. On the other hand, SDPA is performed on known variables within an algorithm in order to deduce further knowledge about the algorithm. The method is formalised through definitions of SDPA vectors, values and matrices.

SDPA vectors could reveal memory addresses, variable values, counters, operations, permutations etc. Cross-iteration comparisons performed through SDPA matrices improve the interpretation of the SDPA values and provide a significant amount of the information required to reverse engineer a secret algorithm.

Clearly, the actual attack would be highly dependent on the algorithm being implemented, the types of countermeasures and the hardware architecture being used, and would require some guesswork on the part of the attacker. In addition to SDPA, supplementary methods like simple power analysis and substitution block attack, would be needed to completely restore the algorithm under attack.

Countermeasures against side-channel attacks may be based on secret code. Such implementations may be the subject of the SDPA as well; however, the efficiency of the method on those algorithms should be studied further. Further research should also be done on operation signatures, in order to capture the information about power bias magnitudes.

The best way to eliminate attack is strict adherence to the cardinal principle. However, the fundamental rule is that the designers must not rely on secrecy of

the algorithm, as the algorithm may be reverse-engineered in the presence of side-channel information leakage.

References

1. Kocher, P.: Timing Attacks on Implementation of Diffie-Hellman, RSA, DSS and Other Systems. In: Koblitz, N. (ed.): Advances in Cryptology - Crypto'96. Lecture Notes in Computer Science, Vol. 1109. Springer-Verlag, Berlin Heidelberg New York (1996) 104–113
2. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.): Advances in Cryptology - Crypto'99. Lecture Notes in Computer Science, Vol. 1666. Springer-Verlag, Berlin Heidelberg New York (1999) 388–397
3. Agrawal D., Archambeault B., Rao J.R., Rohatgi P.: The EM Side-Channel(s): Attacks and Assessment Methodologies. In: Cryptographic Hardware and Embedded Systems - CHES'2002
4. Chari S., Jutla C., Rao J.R., Rohatgi P.: A Cautionary Note Regarding Evaluation of AES Candidates. In: AES Second Candidate Conference, Rome, Italy, March 22–23 (1999) 133–147
5. Quisquater J.J., Samyde D.: Automatic Code Recognition for Smartcards Using a Kohonen Neural Network. Proceedings of the 5th Smart Card Research and Advanced Application Conference - CARDIS'02, San Jose, CA, USA, November 21–22, USENIX Association (2002)
6. Messerges T.S., Dabbish E.A., Sloan R.H.: Examining Smart-Card Security under the Threat of Power Analysis Attacks. IEEE Transactions on Computers, **51(5)** (2002) 541–552
7. Fahn, P.N., Pearson, P.K.: IPA: A New Class of Power Attacks. In: Koc, C.K., Paar, C. (eds.): Cryptographic Hardware and Embedded Systems - CHES'99. Lecture Notes in Computer Science, Vol. 1717. Springer-Verlag, Berlin Heidelberg New York (1999) 173–186
8. Kömmerling, O., Kuhn, M.G.: Design Principles for Tamper-Resistant Smartcard Processors. Proceedings of the USENIX Workshop on Smartcard Technology - Smartcard'99, Chicago, Illinois, May 10–11, USENIX Association (1999) 9–20
9. Chari S., Jutla C.S., Rao J.R., Rohatgi P.: Towards Sound Countermeasures to Counteract Power-Analysis Attacks. In: Wiener, M. (ed.): Advances in Cryptology - Crypto'99. Lecture Notes in Computer Science, Vol. 1666. Springer-Verlag, Berlin Heidelberg New York (1999) 398–412
10. Akkar M.L., Bevan R., Dischamp P., Moyart D.: Power Analysis, What Is Now Possible. In: Okamoto T. (ed.): ASIACRYPT 2000. Lecture Notes in Computer Science, Vol. 1976. Springer-Verlag, Berlin Heidelberg New York (2000) 489–502
11. Clavier C., Coron J.S., Dabbous N.: Differential Power Analysis in the Presence of Hardware Countermeasures. In: Koc C.K., Paar C. (ed.): Cryptographic Hardware and Embedded Systems - CHES'2000. Lecture Notes in Computer Science, Vol. 1965. Springer-Verlag, Berlin Heidelberg New York (2000) 252–263
12. Novak, R.: Side-Channel Attack on Substitution Blocks. In: 1st MiAn International Conference on Applied Cryptography and Network Security - ACNS'2003, Kunming, China. Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (2003) in print
13. Anderson, R., Kuhn, M.: Low Cost Attacks on Tamper Resistant Devices. In: Lomas, M. et al. (ed.): Security Protocols. Lecture Notes in Computer Science, Vol. 1361. Springer-Verlag, Berlin Heidelberg New York (1997) 125–136