Note on side-channel attacks and their countermeasures

In the last few years ciphers making use of table-lookups in large tables—and most notably AES [12, 6]—have received a lot of bad publicity due to their vulnerability to cache attacks [15, 1, 13]. These attacks target the secret key by exploiting the variable time that an AES computation takes due to the storage of the large table in cache. From this one may conclude that algorithms that do not make use of table-lookups are better suited for secure implementations. The subject of this note is to point out that this is in general not the case, more particularly if we consider attacks that exploit measurements of power consumption and electromagnetic radiation. These attacks are relevant if one wants to perform cryptographic computations involving a secret key in a device to which an adversary has physical access. As employees of STMicroelectronics and NXP, we have come across numerous such products and applications. Historically smart cards has been the first commercial product to embed countermeasures against this kind of attacks but this is spreading to RFID, electronic passport and more complex systems. More generally, note that the attacks under consideration do not exploit an inherent weakness of an algorithm, but rather a characteristic of the implementation. For this reason they are called side-channel attacks as opposed to cryptanalytic attacks. Particularly good sources of information on side channel attacks and countermeasures are the proceedings of the yearly CHES conferences (`http://www.chesworkshop.org/`) and the text book [14].

As an example, we consider on the one hand a class of algorithms that do not make use of table-lookups but instead of addition (or subtraction) modulo 2^n, bitwise XOR, shift and rotation. This class of algorithms is denoted by the acronym ARX (Addition-Rotation-XOR) and seems hard to protect against certain side-channel attacks.

On the other hand, algorithms that restrict their choice of operations to bitwise Boolean operations, and (cyclic) shifts can be protected much more efficiently if certain conditions are satisfied. We will illustrate this point with our SHA-3 submission KECCAK[2].

1 Countermeasures against information leakage in power and radiation

If electronic hardware performs computations, its power consumption in general depends on the value of the data being processed. This is the case for CPUs but also for ASICs and FPGA chips. Moreover, any electronic device emits electromagnetic radiations that also depend on the data being processed. The attacks exiting these aspects are denoted by the terms *power analysis* and *electromagnetic analysis*. The general set-up is that the attacker gets one or more traces of the measured power consumption or electromagnetic radiation. If only a single trace suffices to mount an attack, one speaks about simple power analysis (SPA) or simple electromagnetic analysis (SEMA). However, the dependence is typically small and obscured by noise. This can be compensated for by taking many traces, each one representing an execution of the cryptographic primitive with different input values per trace. These many traces are then subject to statistical methods to retrieve the key information. These attacks are called differential power analysis (DPA) [8] and differential electromagnetic analysis (DEMA). An important aspect in these attacks is that the traces must be *aligned*: they must be combined in the time-domain such that corresponding computation steps coincide between the different traces.

In DPA and DEMA one distinguishes between *first order* DPA/DEMA and *higher order* DPA/DEMA. In first-order, the attacker is limited to considering single time offsets of the traces. In m-th order the attacker may incorporate up to m time offsets in the analysis. In this note we focus on protection against first-order DPA/DEMA. Higher-order DPA/DEMA is usually applied to implementations with countermeasures against first-order DPA/DEMA. Higher-order attacks are in principle more powerful but also much harder to implement [14].

In the light of these attacks, one must attempt implementing the cryptographic primitives such that the effort (or cost) of the adversary for retrieving the key is too high for her/him to be interesting. An important countermeasure is implementing the cryptographic primitives such that the power consumption and electromagnetic radiation leak as little as possible on the secret keys or data. Many side-channel attacks have been devised and equally many countermeasures have been published. Countermeasures can be implemented at several levels:

- at the transistor level: logical gates and circuits are built such that the information leakage is reduced;
- at the program level: the order of operations can be randomized or dummy instructions can be inserted randomly to make the alignment of traces more difficult;
- at the algorithmic level: the operations of the cryptographic algorithm are computed in such a way that the information leakage is reduced;
- at protocol level: the protocol is designed such that it limits the number of computations an attacker can provoke with a given key.

As opposed to protection against cryptographic attacks, protection against side channel attacks is never expected to be absolute: a determined attacker with a massive amount of resources will sooner or later be able to break an implementation. The engineering challenge is putting in enough countermeasures such that the attack becomes too expensive to be interesting. Products that offer a high level of security typically implement countermeasures on multiple levels.

The countermeasures at transistor level are independent of the algorithm to be implemented. They imply dedicated hardware for cryptography which takes more area, requires dedicated industrialization processes and is in general more expensive. Moreover, while these countermeasures may significantly reduce the information leakage, there always remains some leakage.

The countermeasures at program level are partially dependent on the algorithm to be implemented. Insertion of dummy instructions is possible in any algorithm while changing the order of operations may be easier for some algorithms than for others. This countermeasure is particularly efficient against higher-order DPA/DEMA as there the signals must be aligned in multiple places and any misalignment severely limits the effectiveness of attack.

The countermeasures at protocol level are also independent of the algorithm. However, what can be done at this level depends on the requirements of the application and in many cases the possibilities are limited.

Finally, the countermeasures at algorithmic level depend on the basic operations used in the algorithm. This is the type of countermeasures where the choice of operations in the cryptographic primitive is relevant. In the remainder of this note we will concentrate on countermeasures at this level.

2 Masking

Masking (sometimes also called blinding) is a countermeasure that offers protection against DPA and DEMA. It consists of representing (part of) the input key and/or data words in a cryptographic primitive by two or more shares (as in secret sharing) where usually the sum or the XOR of the shares is equal to the intended value of the word. Subsequently the program (or circuit) computes the cryptographic primitive using the shares in a way that the intermediate results are never correlated to the actual (unmasked) intermediate values. Whether this is possible depends on the details of the cryptographic primitive and the type of masking. In any case, to achieve decorrelation, for each of the masked variables, all but one of its shares must be generated (pseudo-)randomly for each execution of the cryptographic primitive. Clearly, the generation of the shares, the masking operation of the input words and unmasking operation of output, usually considered out of scope of DPA attacks, must also be carefully implemented to limit information leakage. For masking to be effective, the adversary shall have as little information as possible on the value of the shares.

Taking two shares offers protection against first-order DPA/DEMA. Providing protection against m-th order DPA requires at least $m + 1$ shares. We will concentrate on the case of two shares offering protection against first order DPA/DEMA, but most of our explanation remains valid for the case of more shares. We will denote one share as the *masked variable* and the other as the *mask*.

Performing an operation on a variable in masked form is straightforward if the type of masking is compatible with it. More particularly, if it is a group operation and the masking scheme makes use of the same group operation, associativity permits treating the masked variables and masks separately. If we represent the unmasked variable x by the masked variable x' and the mask r_x such that $x = x' * r_x$, we can compute c' and r_c corresponding with $c = a * b$ as:

$$\begin{aligned} c' &= a' * b' \,, \\ r_c &= r_a * r_b \,. \end{aligned}$$

The two shares c' and r_c representing c can then be used to perform subsequent operations.

In general this technique can be applied if the operation to be protected is linear with respect to the masking scheme. As all separate operations are performed on variables that have no correlation with the actual data a, b and c, this achieves resistance against first order DPA. Note however that the operations on two shares must be *physically isolated* from each other. For example, if a register containing a' is loaded with r_a, the power consumption can depend on the number of bits switched (e.g. on the Hamming weight of $a' \oplus r_a$) and it is likely to leak information on a. This can be solved by setting the register to zero in between. But clearly, care must be taken when attempting to build side-channel resistant implementations.

3 Masking for ARX

For ARX two types of masking operations are appropriate:

- Arithmetic masking: $x = x' + r_x \bmod 2^n$ compatible with addition and subtraction
- Boolean masking: $x = x' \oplus r_x$ compatible with XOR, rotation and shift

When variables are added or subtracted, they must be available in arithmetic masking form and so will be the result; when they are XORed, rotated or shifted, they must be available in Boolean masking form and so will be the result. The problem is now when variables undergo operations of different types; namely when we have a variable in Boolean masking form and it must be added to another variable, or vice versa, when we have a variable in arithmetic masking form and it must be rotated or XORed to another variable.

The only solution to that problem identified up to now is converting a variable in Boolean masking form to arithmetic masking form or vice versa when needed. The number of such conversions required depends on the way the different operations are alternated in the algorithm. This can be quite often as most ARX algorithms get their non-linearity exactly from this alternation.

When performing these conversions, it is equally important not to leak information on the (unmasked) variables being manipulated as in the regular operations. A conversion algorithm is denoted as *secure* if none of the manipulated variables has correlation with the unmasked variable. This problem has been investigated and algorithms are proposed by Jean-Sébastien Coron, Louis Goubin and Alexei Tchulkine in a series of papers published at CHES conferences: [3], [7] and [4]. In the following subsections we summarize the results of these papers. We invite the interested reader to study these papers in detail.

3.1 Boolean to arithmetic masking

In [7] Louis Goubin proposes a secure conversion from Boolean to arithmetic masking that converts a couple (x', r) with $x = x' \oplus r$ to (X', r) with $x = X' + r$. This algorithm takes 7 elementary operations (5 XORs and 2 subtractions), 1 random generation and 2 auxiliary variables.

Note that the computational cost of this conversion is significantly higher than performing an ARX operations on a masked variable.

3.2 Arithmetic to Boolean masking

In [7] Louis Goubin proposed a secure conversion from arithmetic to Boolean masking that converts a couple of n-bit words (X', r) with $x = X' + r$ to (x', r) with $x = x' \oplus r$. The algorithm takes $5(n + 1)$ elementary operations, 1 random generation and 3 auxiliary variables. For $n = 8$, 32 and 64 this implies 45, 165 and 325 elementary operations respectively. Clearly applying this conversion algorithm may slow down an implementation by one or two orders of magnitude.

In [4], Jean-Sébastien Coron and Alexei Tchulkine presented an alternative algorithm achieving the same result but taking less operations. This algorithm makes use of precomputed tables and there is a trade-off in the size of the tables and the number of operations to be performed. In any case, the longer the words, the more operations. For example, for 32-bit words and a processor with 32-bit word length, taking a 16-byte table per mask value r still leads to an arithmetic to Boolean conversion that takes 40 elementary operations. When the words are 64-bit long, doing it in 40 operations on a 64-bit processor requires 512 bytes of table per mask value. Moreover, the cost of computing the tables must be taken into account. While this is certainly an important efficiency improvement with respect to the method without the lookup tables, the overhead remains very important.

Additionally, there is a potential security issue with the use of the same value r during many computations. A conversion table is specific for a given value of the mask r, so just before converting from arithmetic to Boolean masking form, a variable x must be available in the form $x = x' + r$. This, and the fact that the conversion table must be computed in the first place, implies numerous computations with the mask r. In general these computations will leak information that may be used to retrieve the value of r. And once this value is known, the masking loses its effectiveness.

4 Masking for nonlinear bitwise Boolean operations

There are algorithms that can be implemented with only bitwise XOR, (cyclic) shifts, bitwise NOT and nonlinear bitwise Boolean operations such as bitwise AND and OR. For words longer than 1 bit these are not group operations and no compatible masking operation exists. However, one can apply Boolean masking. In [5] we have shown that this can be done easily if the algorithm is put in a form such that the AND (and OR) operations occur in combination with an XOR (with concatenation denoting bitwise AND and $\oplus$ denoting bitwise XOR):

$$a = a \oplus bc \ .$$

Given the Boolean masking form, this can then be computed as follows:

$$\begin{aligned} a' &= a' \oplus b'c' \oplus b'r_c \ , \\ r_a &= r_a \oplus r_b r_c \oplus r_b c' \ . \end{aligned}$$

with the righthand expressions evaluated left-to-right. It follows that the single XOR and single AND in the plain implementation grows to 4 XORs and 4 ANDs in the masked implementation. A similar argument applies for the OR. This generalizes nicely to protection against higher-order DPA/DEMA. We refer to [5] for some more discussion on this and the distinguishing power of second order DPA as compared to first order DPA.

4.1 The example of KECCAK

KECCAK can be implemented with only bitwise XORs, AND and NOT operations and rotations. The nonlinear operations in KECCAK are the same as those in BASEKING, the cipher that is the subject of [5] and hence the techniques shown there can be readily applied. Masking doubles the number of XOR and rotation operations and multiplies the number of combined XOR/AND operations by four. From this it follows that masking KECCAK triples the number of bitwise Boolean operations and doubles the number of rotations.

5 Dedicated hardware implementations

Implementing an algorithm in hardware such that it offers protection against side-channel attacks raises even more concerns. The masking conversion problem for ARX remains a problem. One approach could be to implement the conversion algorithms in hardware. In doing this one should take care that the order in which the operations are performed in the conversion algorithms are respected. More particularly, the presence of glitches may result in an information leakage that is not modeled by the masking theory and a masked implementation may become vulnerable [9, 10].

This concern also exists for algorithms that only make use of bitwise Boolean operations and (cyclic) shifts. Recently Svetla Nikova, Vincent Rijmen and Martin Schäffler presented in [11] a masking technique that offers resistance against side-channel attacks even in the presence of glitches. This technique consists of representing the data elements with three (or more) shares and makes sure that in any computation at least one of the share is not taken as input. This technique is well suited for algorithms in which the nonlinear components can be expressed as nonlinear Boolean functions with at least 3, but preferably no more, variable. As such it is very well suited for KECCAK, with its single nonlinear gate per bit in the round function. In ARX algorithms the only nonlinear operations (in GF(2)) are the additions. One may consider implementing them in the simplest bitwise way, with the bits of $c = a + b$ computed as $c_i = a_i \oplus b_i \oplus d_i$ with d_i the carry bit. This is a linear expression and poses no problem for implementing in masked form. The computation of the carry bit is given by the majority function of a_{i-1}, b_{i-1} and d_{i-1} and is also suited to be implemented with the techniques proposed in [11]. The disadvantage is that with this choice the computation of the bits of c is done in a serial way, leading to large gate delays.

6 Conclusions

In this note we have addressed some issues that arise when trying to implement cryptographic primitives that should offer resistance against side channel attacks. Although they do not make use of table-lookups, ARX algorithms may be hard or expensive to protect against side channel attacks, while this appears to be significantly easier for algorithms that only make use of bitwise Boolean operations and (cyclic) shifts.

The KECCAK Team, May 2009
Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche

References

[1] D. J. Bernstein, *Cache-timing attacks on AES*, 2005, Document ID: cd9faae9bd5308c440df50fc26a517b4, `http://cr.yp.to/papers.html#cachetiming`.

[2] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, KECCAK *specifications*, NIST SHA-3 Submission, October 2008, `http://keccak.noekeon.org/`.

[3] J. Coron and L. Goubin, *On Boolean and arithmetic masking against differential power analysis*, CHES (Ç. K. Koç and C. Paar, eds.), Lecture Notes in Computer Science, vol. 1965, Springer, 2000, pp. 231–237.

[4] J. Coron and A. Tchulkine, *A new algorithm for switching from arithmetic to Boolean masking*, in Walter et al. [16], pp. 89–97.

[5] J. Daemen, M. Peeters, and G. Van Assche, *Bitslice ciphers and power analysis attacks*, Fast Software Encryption 2000 (B. Schneier, ed.), Lecture Notes in Computer Science, vol. 1978, Springer, 2000, pp. 134–149.

[6] J. Daemen and V. Rijmen, *The design of Rijndael — AES, the advanced encryption standard*, Springer-Verlag, 2002.

[7] L. Goubin, *A sound method for switching between Boolean and arithmetic masking*, CHES (Ç. K. Koç, D. Naccache, and C. Paar, eds.), Lecture Notes in Computer Science, vol. 2162, Springer, 2001, pp. 3–15.

[8] P. C. Kocher, J. Jaffe, and B. Jun, *Differential power analysis*, CRYPTO (M. J. Wiener, ed.), Lecture Notes in Computer Science, vol. 1666, Springer, 1999, pp. 388–397.

[9] S. Mangard, T. Popp, and B. M. Gammel, *Side-channel leakage of masked CMOS gates*, CT-RSA (A. Menezes, ed.), Lecture Notes in Computer Science, vol. 3376, Springer, 2005, pp. 351–365.

[10] S. Mangard, N. Pramstaller, and E. Oswald, *Successfully attacking masked AES hardware implementations*, CHES (J.R. Rao and B. Sunar, eds.), Lecture Notes in Computer Science, vol. 3659, Springer, 2005, pp. 157–171.

[11] S. Nikova, V. Rijmen, and M. Schläffer, *Secure hardware implementation of non-linear functions in the presence of glitches*, ICISC (P. J. Lee and J. H. Cheon, eds.), Lecture Notes in Computer Science, vol. 5461, Springer, 2008, pp. 218–234.

[12] NIST, *Federal information processing standard 197, advanced encryption standard (AES)*, November 2001.

[13] D. A. Osvik, A. Shamir, and E. Tromer, *Cache attacks and countermeasures: The case of AES*, CT-RSA (D. Pointcheval, ed.), Lecture Notes in Computer Science, vol. 3860, Springer, 2006, pp. 1–20.

[14] E. Oswald S. Mangard and T. Popp, *Power analysis attacks — revealing the secrets of smartcards*, Springer-Verlag, 2007.

[15] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, *Cryptanalysis of DES implemented on computers with cache*, in Walter et al. [16], pp. 62–76.

[16] C. D. Walter, Ç. K. Koç, and C. Paar (eds.), *Cryptographic hardware and embedded systems - ches 2003, 5th international workshop, cologne, germany, september 8-10, 2003, proceedings*, Lecture Notes in Computer Science, vol. 2779, Springer, 2003.