Efficient Use of Random Delays

Olivier Benoit[1] and Michael Tunstall[2]

[1] Gemalto *olivier.banoit@gemalto.com*
[2] Royal Holloway, University of London *m.j.tunstall@rhul.ac.uk*

Abstract

Random delays are commonly used as a countermeasure to inhibit side channel analysis and fault attacks in embedded devices. This paper proposes a different manner of generating random delays. The alternative proposed increases the desynchronisation compared to uniformly distributed random delays. It is also shown that it is possible to reduce the amount of time lost due to random delays, while maintaining the increased variation.

1 Introduction

The use of random delays in embedded software is often proposed as a generic countermeasure against side channel analysis, such as simple power analysis (SPA) (e.g. [2]) or statistical analysis of the current consumption [6], or electromagnetic emanations [4]. Statistical analysis is meaningless in presence of desynchronisation; an attacker must resynchronise the acquisitions at the area of interest before being able to interpret what is happening at a given point in time. The specific case using random delays to defend against cache based side channel analysis is discussed in [7].

This effect is discussed in detail in [3], where the case of hardware random delays is considered. This involves random clock cycles being added into the process to create desynchonisation is considered. An attack based on integrating adjacent clock cycles to find the required information to conduct a Differential Power Analysis [6] points. In this paper the case where a software delay is introduced at various points in a process to introduce desynchonisation. This will introduce a delay in the process that is too large to allow a similar attack to be conducted. The attacks described in [3] require the acquisitions to be locally synchonised.

It has also been proposed as a countermeasure against fault attacks that require a high degree of precision in where the fault is injected [1]. A series of random delays will make it difficult to implement these attacks as the target point in time is constantly changing its position. An attacker is obliged to attack the same point numerous times and to wait until the fault and the target coincide. Another alternative consists of applying a realtime synchronisation on the target event, but this technique is difficult to use and, once again, adds another step to the attack. The events that an attacker can try to trigger a fault attack are also limited so the use of random delays during a sensitive process will still cause problems.

In both cases the greater the variation caused by the random delay, the more difficult it is to overcome. Nevertheless, the use of random delays is problematic as it serves no purpose other than to desynchronise events. It cannot prevent an attack, it just renders the attack more complex.

Software random delays are generally uniformly distributed. A change in the distribution of the random used is proposed in order to improve the efficiency of the random delay after several individual random delays are cumulated. This involves changing the distribution of the random values so that they are no longer distributed over a uniform distribution. The ideal criteria for the new distribution being the following:

1. Random delays should provide more variation when compared to the same number of random delays generated from uniformly distributed random variables.

2. The overall performance of the smart card should be similar or better than if the random delays are distributed over a uniform distribution.

3. Individual random delays should not make an attack easier in the event there is only one random delay between the synchronisation point and the target process.

4. It should be a non-trivial task to derive the distribution of the random delay used. If a given delay is known to be more common than another it may be possible to base an attack on this knowledge.

These conditions are the ideal; there will be some compromise needed between all of the conditions as the uniform distribution will probably be preferable in some situations.

Our main aim in this paper is to optimise conditions 1 and 2 since the reverse engineering of a random delay distribution is rarely conducted (Furthermore, it will be shown that this involves more effort than assuming the distribution is uniform) and individual efficiency is rarely applicable due to the numerous random delays used in a sensitive process.

A practical example of how these distributions can be designed is given. This will need to be adapted for a given situation. In cryptographic algorithms, for example, it is prudent to have lots of small random delays so that someone conducting a side channel attack can only synchronise their acquisitions locally. Whereas, in an operating system the random delays can be larger as sensitive areas will be further apart.

In order to be able to compare different distributions the mean and the standard deviation are used to express the characteristics of the sum distribution of random delays. This was a natural choice as the sum distribution is based on a binomial distribution, which is the discrete version of a normal distribution (characterised by the mean and variance). The standard deviation was chosen rather than the variance as this represents the mean deviation from the mean.

The paper is organised as follows. Section 2 details the usual case for choosing a random value for generating a random delay and the cumulative effect. Section 3 details how this can be modified to increase both security and performance. Section 4 describes the effect of these modifications on one random delay and how this system could potentially be reverse engineered. This is followed by the conclusion in Section 5.

2 The Discrete Uniform Distribution

The uniform distribution is the usual distribution of choice for generating a random delay. This provides the best protection between two points if one random delay is present between them. Subsequent random delays will provide a cumulative random delay as shown in Figure 1.

As can be seen after several random delays the distribution rapidly becomes binomial in nature i.e. a discrete normal distribution. It is interesting to note that after 10 such random delays there is a tail either side of the binomial where the probability of a delay of this length occurring is negligible.

3 Modifying the Probability Function

In order to increase the variation in the cumulative distribution the probability of the extreme values occurring i.e. the minimum and maximum values of one random delay. The formation of a binomial distribution after several random delays cannot be avoided, as the number of combinations close to the mean value are far too large. Any modification is also required to be subtle, as one random delay needs to be able to provide desynchronisation between two sensitive areas (criteria 2). It was initially assumed that this could be achieved by a distribution of the form shown in Figure 2.

To use this probability function in a constrained environment, for example a smart card, this will need to be expressed as a table where the number of entries for a given value represent the probability of that value being chosen. A uniformly distributed random number can then be used to select an entry from this table. The function that was chosen governs the amount of entries for each value of x, where x can take integer values in the interval $[0, N]$:

$$y = \lceil ak^x + bk^{N-x} \rceil$$

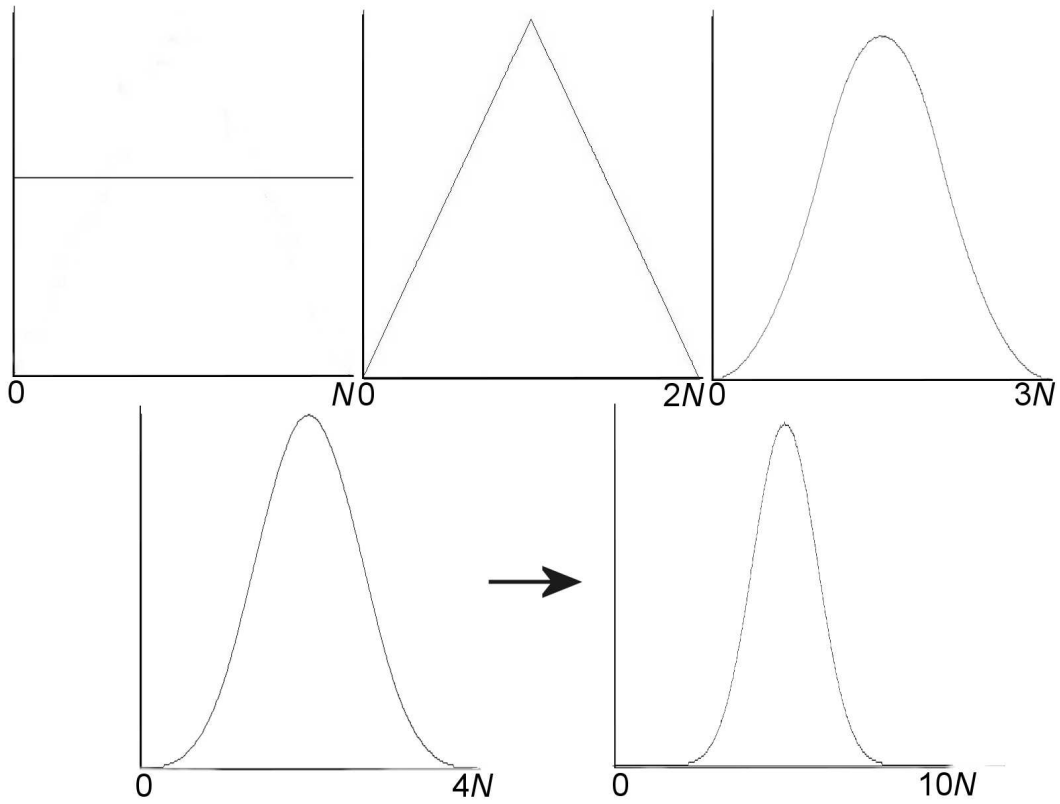

Figure 1. The cumulative random delay generated by a sequence of random delays generated from uniformly distributed random variables. The amount of random delays considered are 1, 2, 3, 4, and 10 from top left to bottom right.

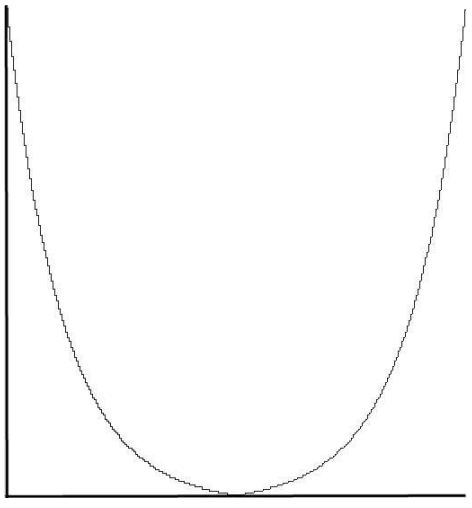

Figure 2. An example of a modified probability function.

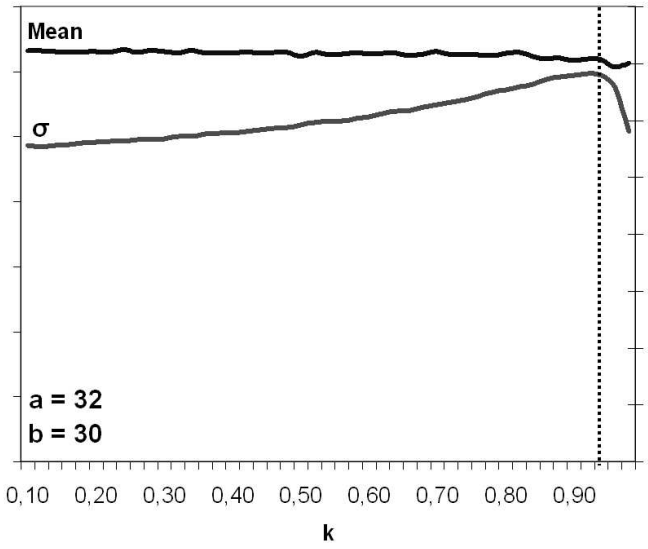

Figure 3. The mean and the standard deviation against k for approximately equal values of a and b.

Where a and b represent the values of the probability function when x takes 0 and N respectively, and y governs the number of entries in a table for each value of x that can be used to represent this function. The sum of y for all values of x will therefore give the total number of table entries. k governs the slope of the curve and can take values in the interval $(0, 1)$. Different values for a and b are used so that a bias can be introduced into the sum distribution to lower the mean delay. The two elements ak^x and bk^{N-x} are both close to zero when $x \approx N/2$ for the majority of values of k that will be of interest. The ceiling function is therefore used to provide a minimum number of entries in the table for each value. This is important as if values are removed from the distribution it will decrease the amount of values the random delay can take. This will make it easier to detect that the random delay used is not uniformly distributed.

The parameters that generate this curve shape were changed and the effect on the sum distribution was tested to search for the best configuration that would satisfy the criteria described, where the values that the random delay could take, and therefore x, are the integer values in the interval $[0, 255]$. For values of a and b that are approximately equal the change in k will have an effect on the mean and the standard deviation as shown in Figure 3. This graph was generated by analysing a large number of random values generated by a random look-up on a table, as described above.

The mean in Figure 3 is fairly constant for all the values of k tested. The variance shows an optimal value of $k = 0.92$. This experiment was repeated for various different values of a and b and the optimal value for k remains constant.

This value was used to look at the effect of varying b on the mean and the standard deviation, as shown in Figure 4. As can be seen the mean can be reduced by an asymmetric distribution that favours the lower extreme over the higher extreme values. The mean shows an almost linear relationship with b, and the standard deviation has a logarithmic relationship with b. The best configuration would be to minimise the mean value and maximise the standard deviation. This is not possible and a compromise needs to be found between the two. $b = 16$ seemed to be a good compromise that was used to conduct further investigation.

To provide a table that can be efficiently implemented the number of entries in the table should be a power of 2. A random number generator will provide a random word where the relevant number of bits can be masked off and used to index the table dictating the length of the random delays. If the number of entries is not equal to a power of two any values generated between the number of entries and the next power of two will have to be discarded. This testing of values will slow the command down and have a potentially undesirable effect on the distribution of the random delays as suitable random numbers will only be regenerated with a certain probability.

The parameters that were found to naturally generate a table of 2^9 entries are shown in Table 1. It would

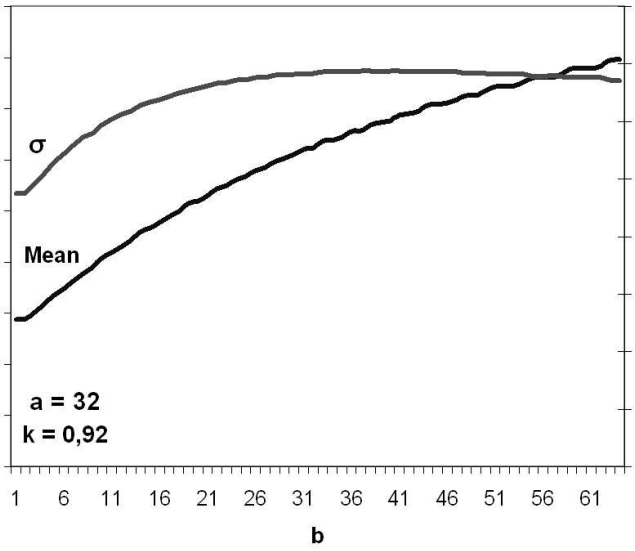

Figure 4. The mean and the standard deviation against b for fixed values of a and k.

Table 1. Parameter Characteristics for Tables of 2^9 Entries

a	b	k	Mean % decrease	σ % increase
22	13	0.88	13.4	34.5
23	12	0.88	16.3	33.5
23	15	0.87	11.6	35.4
24	11	0.88	19.7	32.1
24	14	0.87	13.4	34.9
25	10	0.88	22.6	30.7
26	6	0.89	32.8	23.5
26	9	0.88	25.6	29.0
26	12	0.87	19.7	32.5
32	8	0.86	31.4	25.8

be possible to choose some parameters and then modify the table so that it has 2^9 entries, but this was deemed overly complicated as this occurs naturally. The percentage changes shown are in comparison to the mean and standard deviation of a uniformly distributed random delay. The change in the mean and the standard deviation is independent to the number of random delays that are added together.

As can be seen a large variation is visible in the change in the mean and standard deviation. It can also be seen that we cannot achieve the best standard deviation increase and mean decrease at the same time. The designer will have to make a compromise between maximising the standard deviation or minimising the mean.

Table 2 shows the parameters that naturally generate a table of 2^{10} entries. The effect of the modified random delays is more pronounced when based on a table of 2^{10} entries, as it is possible to approach the optimal value of 0.92 for k. However, it may not be realistic to have a table of 1024 in a constrained environment such as a smart card, although in modern smart cards the problem of lack of code memory is starting to disappear. An example of the effect of using this sort of method to govern the length of random delays is shown in Figure 5, where $a = 26$, $b = 12$ and $k = 0.87$.

This distribution satisfies the criteria 1 and 2 as described in the introduction. It is unlikely to satisfy criteria 3 as the distribution does have some undesirable properties. The probability of a 0 being produced by the table is

Table 2. Parameter Characteristics for Tables of 2^{10} Entries

a	b	k	Mean % decrease	σ % increase
33	18	0.94	21.3	39.2
37	14	0.94	32.2	32.7
50	32	0.90	16.2	46.5
53	29	0.90	21.3	44.6
54	28	0.90	23.3	43.6
55	19	0.91	35.8	34.7
56	34	0.89	18.2	46.6
58	32	0.89	21.5	45.3
59	23	0.90	32.3	38.4
60	22	0.90	34.3	36.9

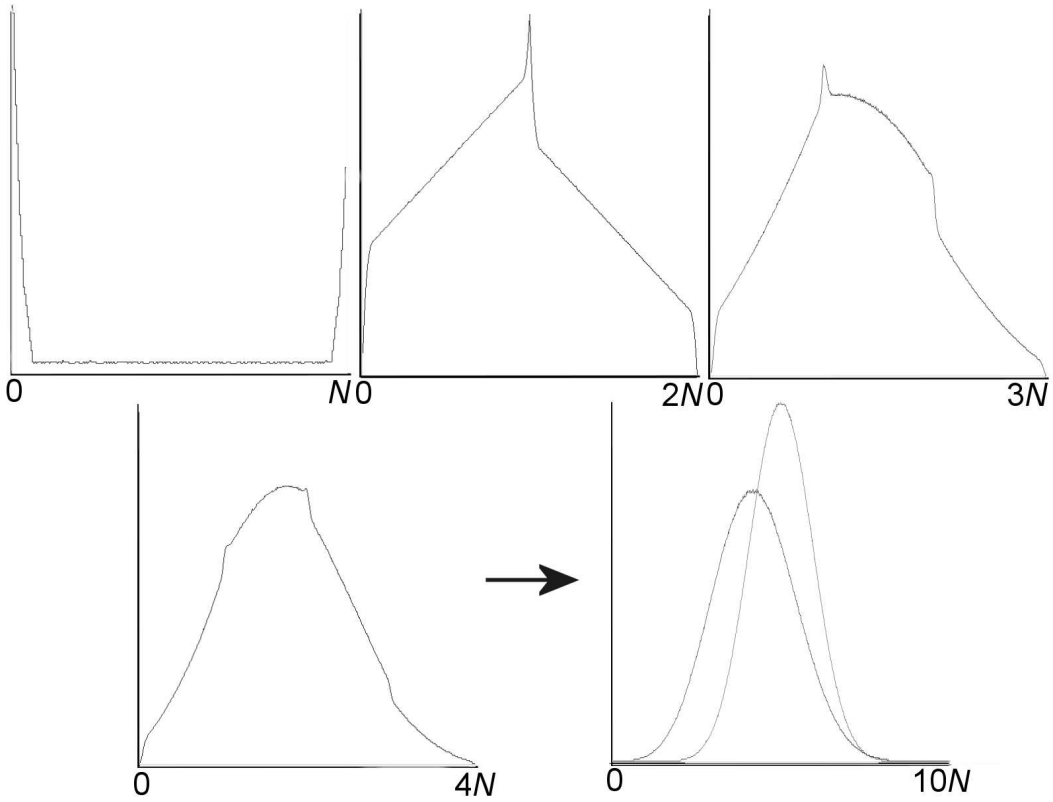

Figure 5. The cumulative random delay generated by a sequence of random delays generated from the biased distribution shown in the top left of the Figure random variables. The amount of random delays considered are 1, 2, 3, 4, and 10 from top left to bottom right. The last graph also shows the distribution of 10 uniformly distributed random delays for comparative purposes.

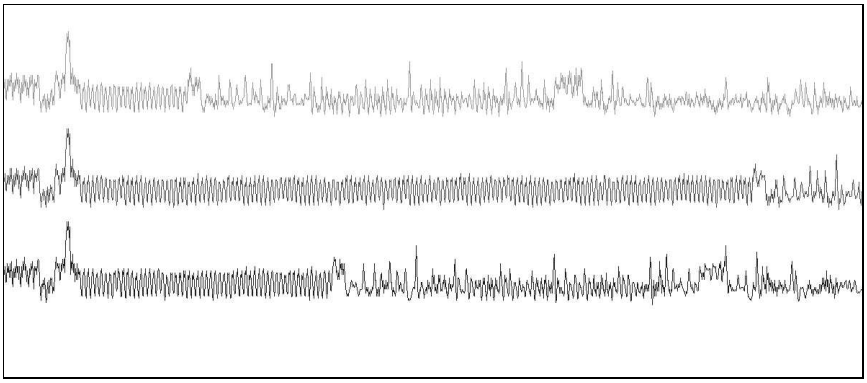

Figure 6. The random delays can be visible in the change in current over time.

$a + 1$ over the number of entries in the table. If a set of acquisitions are locally synchronised, a larger number of acquisitions will remain synchronised after the first random delay than would be expected if the random delay was uniformly distributed. This is an undesirable property if one random delay is separating the synchronisation point from the target. It is therefore still advisable to use a uniform distribution after the most obvious synchronisation point. Nevertheless, the described distribution will make the task of *a posteriori* synchronisation more difficult.

4 Reverse Engineering the Distribution

In order to evaluate whether a given probability function will fulfill criteria 4 it is necessary to describe the process that would need to be undertaken to derive the distribution being used. A series of current acquisitions would need to be taken and locally resynchronised. The first random delay that occurs after this point can then be studied.

An example of the acquisitions that could be produced is given in Figure 6. The power consumption is synchronised on the left hand side of the image. A repeating pattern can then be observed that occurs in each acquisition but last for a variable length of time. The power consumptions traces are then desynchronised on the right hand side of the image.

A reasonable amount of data will need to be collected and the lengths of the random delays stored. These values can then be tested statistically to determine whether or not they correspond to a uniform distribution or not. This can be done by conducting a χ^2 test on the acquired data. An attacker will most likely be required to measure each delay by hand, which will be a lengthy and tedious process. It would be possible to develop a tool for a given chip that would generate this information, but as soon as the chip is changed the tool would need to be updated as different chips change the current in different ways.

This process can be quite complex if hardware random delays are also present. These normally take the form of randomly inserted clock cycles where the chip will not do anything for 1 clock cycle or the form of an unstable internal clock. This is likely to add serious complexity to the resynchronisation process. These effects are ignored for our analysis but will make an attacker work much harder for the desired information.

It can be shown that the random delay being observed is not based on a uniformly distributed random value by using a χ^2 test with a null hypothesis that the random delays are uniformly distributed. In order to conduct this test the minimum frequency threshold should be around $5N$ (a rule of thumb given in [5]). In the example, given the minimum frequency threshold, 1280 samples would be required. In practice, far fewer samples are required to consistently provide evidence against the null hypothesis when one of the proposed distributions are used. However, the attacker cannot trust these results without repeating the test with independent acquisitions, so the actual amount of data acquired will probably still need to be around $5N$.

It is only necessary for an attacker to determine the presence of a bias to try and base an attack on the changed distribution. The attacker can readily show with the χ^2 test that the random delay observed is not uniformly distributed. This process may be of use if an attacker is trying to mount a dynamic attack where several faults need to be injected into a process protected by random delays. The distribution of the random delays can be

characterised to determine the best configuration that will be able to circumvent the protection. If only one random delay is present between two vulnerable areas the uniform distribution is the optimal defence. If several random delays are present the solution presented is a better alternative. In the case of power attacks, where the interpretation is done *a posteriori*, the above analysis is unnecessary. The simplest solution is to locally synchronise the acquisitions either side of the random delay, as would be done with a uniformly distributed random delay.

5 Conclusion

In this paper we have demonstrated that the standard deviation of cumulated random delays can be improved upon by using a specific distribution for each individual random delay. It has also been shown that the expected amount of time lost due to random delays can be reduced to minimise the impact of this countermeasure on the performance of functions they are protecting. This is not presented as an optimal solution, but is assumed to be close to optimal.

There are two strategies that can be adopted to alleviate the fact that criteria 3 will never be satisfied by this strategy.

1. The use of the uniform distribution can be used where the security of one random delay is paramount, e.g. a random delay between the modification of two separate key bytes, and use the modified distribution elsewhere.

2. Always use the modified distribution. It has been shown that the amount of work required to conduct this analysis is far greater than to the extra work caused by the modified distribution. If an attacker knows the modified distribution has been used the amount of work to remove the desynchronisation is reduced. However, if the distribution is unknown there is no reason to go through a characterisation process when the algorithms that would work against a system using uniformly distributed random delays will work against systems using the proposed method of generating random delays, albeit more slowly.

This represents an unusual situation, where a method of increasing the security of a process can also reduce the amount of computational time required. Normally, in smart card cryptographic algorithm implementations, the addition of more secure features always comes with a reduction in performance.

References

[1] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerers apprentice guide to fault attacks. *Proceedings of the IEEE Special Issue on Cryptography and Security*, 94(2):370–382, 2006.

[2] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo. AES power attack based on induced cache miss and countermeasures. In *International Symposium on Information Technology: Coding and Computing — ITCC 2005*, pages 586–591. IEEE Computer Society, 2005.

[3] C. Clavier, J.-S. Coron, and N. Dabbous. Differential power analysis in the presence of hardware countermeasures. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer-Verlag, 2000.

[4] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer-Verlag, 2001.

[5] D. Knuth. *The Art of Computer Programming*, volume 2, Seminumerical Algorithms. Addison–Wesley, third edition, 2001.

[6] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.

[7] D. Page. Defending against cache based side-channel attacks. *Information Security Technical Report*, 8(1):30–44, April 2003.