Analysis and Improvement of the Random Delay Countermeasure of CHES 2009

Jean-Sébastien Coron and Ilya Kizhvatov

Université du Luxembourg
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
{jean-sebastien.coron, ilya.kizhvatov}@uni.lu

Abstract. Random delays are often inserted in embedded software to protect against side-channel and fault attacks. At CHES 2009 a new method for generation of random delays was described that increases the attacker's uncertainty about the position of sensitive operations. In this paper we show that the CHES 2009 method is less secure than claimed. We describe an improved method for random delay generation which does not suffer from the same security weakness. We also show that the paper's criterion to measure the security of random delays can be misleading, so we introduce a new criterion for random delays which is directly connected to the number of acquisitions required to break an implementation. We mount a power analysis attack against an 8-bit implementation of the improved method verifying its higher security in practice.

Keywords: Side channel attacks, DPA, countermeasures, random delays

1 Introduction

Embedded software implementations of cryptographic algorithms are threatened by physical attacks like Differential Power Analysis (DPA) or fault injection. The simplest method of protection against such attacks consists in randomizing the flow of the operations by shuffling the order of the operations or inserting random delays composed of dummy operations. These *hiding* countermeasures offer less security than *masking* countermeasures but have smaller implementation and performance costs. For the general background on physical attacks and countermeasures we refer the reader to the book [6].

Random delays in software. Software random delays are implemented as loops of "dummy" operations that are inserted in the execution flow of an algorithm being protected. A single delay can be removed relatively easy by static alignment of side-channel traces, *e.g.* with cross-correlation techniques [4]. Therefore, the execution should be interleaved with delays in multiple places. To minimize the performance overhead in this setting, individual delays should be

possibly shorter. An attacker would typically face a cumulative sum of the delays between the synchronization point (which would usually be at the start or at the end of the execution) and the target event. So the cumulative delay should increase the uncertainty of an attacker about the location of the target event in time. For further discussion on random delays in software we refer the reader to [3] and [8]. We also note that elastic alignment techniques were reported in [9] to be able to reduce the effect of multiple delays. Within the scope of this paper we do not verify these techniques, assuming an attacker without elastic alignment.

Previous work. An efficient method for random delay generation in embedded software was suggested in CHES 2009 [3] under the name of Floating Mean. The central idea of the method is to generate the delays non-independently within one run of a protected algorithm. In this way, the adversary facing the effect of the cumulative sum of the delays will have to cope with much larger variance and thus will require significantly more side channel measurements compared to other methods with independently generated random delays such as [8]. We recall the Floating Mean method in Sect. 2.

Our contributions. Here we discover that the Floating Mean method of [3] can suffer from an improper parameter choice, offering less security than expected. We perform a detailed statistical analysis of the Floating Mean method of [3] and we show how to choose correct parameters (see Sect. 3).

However these new parameters require longer delays, which means the number of delays should be relatively small to keep a reasonable performance overhead. This is not good for security because, as discussed above, in general few long delays are easier to detect and remove than multiple short delays. Therefore we propose an improved method for random delay generation which can work with short delays (Sect. 4). Our new method is easy to implement; we describe a concrete implementation in assembly language (Appendix B).

We also show that the criterion in [3] to measure the efficiency of random delays can be misleading and derive a new efficiency criterion that is information-theoretically sound (Sect. 5). Finally, we mount a practical DPA attack against the implementation of our improved method on an 8-bit AVR microcontroller to verify its higher security. With these results, we target practical designers who implement the timing randomization countermeasures for protection of their embedded software implementations.

2 The Floating Mean Method

Here we recall the Floating Mean method introduced in [3]. Most methods for random delay generation use independent individual delays; in this way, the cumulative sum of many delays which an adversary is facing in an attack tends to normal distribution with the variance being the sum of variances of the individual delays. Instead the core idea of the Floating Mean method from [3] is to

generate random delays non-independently, in order to increase the variance of the cumulative sum. More precisely, the delays are generated as follows:

1. Initially the integer parameters a and b are chosen so that $b < a$, where a determines the worst-case delay and b determines the variance of the delays within a single run of a protected algorithm;
2. Before each run of a protected algorithm, a integer value m is generated independently and uniformly on $[0, a - b]$;
3. Within each run, the integer lengths of individual delays are generated independently and uniformly on $[m, m + b]$.

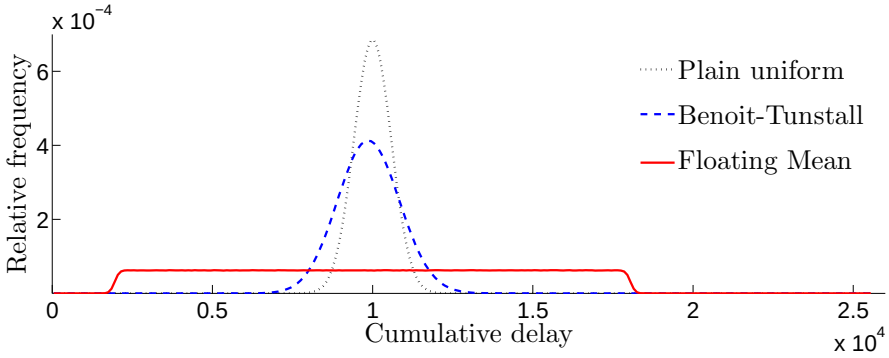


Fig. 1. Empirical distributions of the sum of 100 delays for random delay generation algorithms, for the case of equal means (based on [3]; delay length counted in atomic delay units)

As shown in [3] the variance of the cumulative sum of the delays in an execution becomes quadratic in the number N of delays instead of linear when the delays are generated independently. This is a significant improvement over the plain independent uniform delays or the table-based method of Benoit and Tunstall [8]. An illustrative example of the distribution of the cumulative sum of 100 delays for the Floating Mean compared to other methods is illustrated by Figure 1 based on [3]. The parameters for the Floating Mean here are $a = 200$, $b = 40$; the parameters for other methods are chosen so that all the methods yield the same mean cumulative delay.

However in practice it is better to have many short random delays rather than long delays, as recommended in [3]; this is because it is more complex to distinguish and remove many short delays than just few long delays. Therefore under this recommendation one should choose smaller values of a and b, for example $a = 18$ and $b = 3$ as used in [3] for the practical implementation. However in the next section we show that for such range of parameters the Floating Mean method provides less security than expected.

3 The Real Behavior of Floating Mean

In this section, we show that the Floating Mean method from [3] provides less security than expected for small a and b.

We begin with taking a detailed look at the distributions of different methods by simulating them with the exact experimental parameters used in the implementation of [3]. In Figure 2 we present histograms of the distributions for different methods. Namely, the number of delays in the sum is $N = 32$ and the parameters of the Floating Mean method are $a = 18$, $b = 3$. The histograms present the relative frequency of the cumulative delay against its duration[1]. We

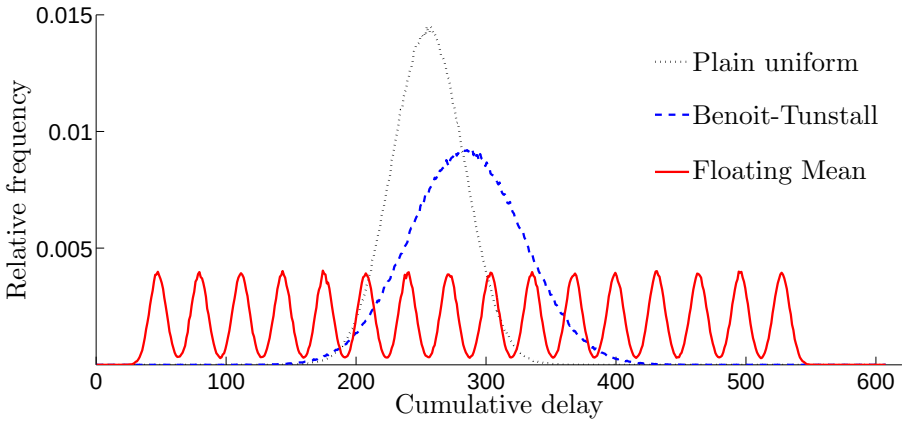


Fig. 2. Empirical distributions of the sum of 32 delays in the experiment of [3]

clearly see a multimodal distribution for the Floating Mean method: the histogram has a distinct shape of a saw with 16 cogs, and not a flat plateau as one would expect from [3].

These cogs are not good for security since they make it easier for an attacker to mount an attack. The classical DPA will be more efficient since the signal is concentrated on the top of the 16 cogs instead of being spread over the clock cycles. In case of an attack with windowing and integration [2], the attacker would integrate the values around cog maximums, omit the minimums to reduce the noise (assuming noise is the same in all the points of the trace) and thus gain a reduction in the number of traces required for an attack.

[1] As in Figure 1, the duration in Figure 2 is expressed in atomic delay units, *i.e.* as the total number of delay loop iterations. To obtain the value in clock cycles one should multiply this by the length of a single delay loop, which is 3 clock cycles in the implementation of [3].

3.1 Explaining the Cogs

Here we explain how cogs arise in the distribution of the Floating mean and we show how to choose the parameters to avoid the cogs.

The distribution for the Floating Mean is in fact a *finite mixture* [7] of $a-b+1$ components with equal weights. Every component corresponds to a given integer value m in $[0, a-b]$. For a given m, the cumulative sum of random delay durations is the sum of N random variables uniformly and independently distributed in $[m, m+b]$. Therefore it can be approximated by a Gaussian distribution with mean

$$\mu_m = N \cdot (m + b/2)$$

and variance

$$V = N\frac{(b+1)^2 - 1}{12} .$$

Therefore the probability density of the distribution for random integer $m \in [0, a-b]$ can be approximated by:

$$f(x) = \sum_{m=0}^{a-b} \frac{1}{(a-b+1)\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu_m)^2}{2\sigma^2}\right)$$

where all components have the same standard deviation $\sigma = \sqrt{V}$:

$$\sigma = \sqrt{N} \cdot \sqrt{\frac{(b+1)^2 - 1}{12}} .$$

The cog peaks are the modes of the components, located in their means μ_m. The distance between the means of successive components is $\mu_{m+1} - \mu_m = N$. We can consider the cogs distinguishable by comparing the standard deviation σ of the components to the distance N between the means. Namely, the distribution becomes multimodal whenever $\sigma \ll N$. In the case of practical implementation in [3], we have $a = 18$, $b = 3$ and $N = 32$, which gives $\sigma = 6.3$; therefore we have $\sigma < N$ which explains why the 16 cogs are clearly distinguishable in the distribution in Figure 2. However for $a = 200$, $b = 40$ and $N = 100$ we get $\sigma = 118$ so $\sigma > N$ which explains why the cogs are indistinguishable in Figure 1 and a flat plateau is observed instead.

3.2 Choosing Correct Parameters

From the above we derive the simple rule of thumb for choosing Floating Mean parameters. To ensure that no distinct cogs arise, parameter b should be chosen such that $\sigma \gg N$. For sufficiently large b we can approximate σ by:

$$\sigma \simeq \frac{\sqrt{3}}{6} \cdot b \cdot \sqrt{N}$$

Therefore this gives the condition:

$$b \gg \sqrt{N}\,. \tag{1}$$

However as observed in [3] it is better to have a large number of short random delays rather than a small number of long delays; this is because rare longer delays are a priori easier to detect and remove than multiple short delays. But we see that condition (1) for Floating Mean requires longer delays since by definition the length of random delays is between $[m, m+b]$ with $m \in [0, a-b]$. In other words, condition (1) contradicts the requirement of having many short random delays.

In the next section we describe a variant of the Floating Mean which does not suffer from this contradiction, *i.e.* we show how to get short individual random delays without having the cogs in the cumulative sum.

4 Improved Floating Mean

In the original Floating Mean method an integer m is selected at random in $[0, a-b]$ before each new execution and the length of individual delays is then a random integer in $[m, m+b]$. The core idea of the new method is to improve the granularity of random delay generation by using a wider distribution for m. More precisely the new method works as follows:

1. Initially the integer parameters a and b are chosen so that $b < a$; additionally we generate a non-negative integer parameter k.
2. Prior to each execution, we generate an integer value m' in the interval $[0, (a-b) \cdot 2^k[$.
3. Throughout the execution, the integer length d of an individual delay is obtained by first generating a random integer $d' \in [m', m' + (b+1) \cdot 2^k[$ and then letting $d \leftarrow \lfloor d' \cdot 2^{-k} \rfloor$.

4.1 Analysis

We see that as in the original Floating Mean method, the length of individual delays is in the interval $[0, a]$. Moreover if the integer m' generated at step 2 is such that $m' = 0 \mod 2^k$, then we can write $m' = m \cdot 2^k$ and the length d of individual delays is uniformly distributed in $[m, m+b]$ as in the original Floating Mean method.

When $m' \neq 0 \mod 2^k$, writing $m' = m \cdot 2^k + u$ with $0 \leq u < 2^k$, the delay's length d is distributed in the interval $[m, m+b+1]$ with a slightly non-uniform distribution:

$$\Pr[d = i] = \begin{cases} \frac{1}{b+1} \cdot (1 - u \cdot 2^{-k}) & \text{for } i = m \\ \frac{1}{b+1} & \text{for } m+1 \leq i \leq m+b \\ \frac{1}{b+1} \cdot u \cdot 2^{-k} & \text{for } i = m+b+1 \end{cases}$$

Therefore when m' increases from $m \cdot 2^k$ to $(m+1) \cdot 2^k$ the distribution of the delay length d moves progressively from uniform in $[m, m+b]$ to uniform in $[m+1, m+b+1]$. In Appendix A we show that for a fixed m':

$$\mathrm{E}[d] = m' \cdot 2^{-k} + \frac{b}{2}$$
$$\mathrm{Var}[d] = \mathrm{E}[d^2] - \mathrm{E}[d]^2 = \frac{(b+1)^2 - 1}{12} + 2^{-k} \cdot u \cdot (1 - 2^{-k} \cdot u)$$

where $u = m' \mod 2^k$.

For a fixed m' the cumulative sum of random delay durations is the sum of N independently distributed random variables. Therefore it can be approximated by a Gaussian distribution with mean:

$$\mu_{m'} = N \cdot \mathrm{E}[d] = N \cdot \left(m' \cdot 2^{-k} + \frac{b}{2} \right) \tag{2}$$

and variance

$$V_{m'} = N \cdot \mathrm{Var}[d] = N \cdot \left(\frac{(b+1)^2 - 1}{12} + 2^{-k} \cdot u \cdot (1 - 2^{-k} \cdot u) \right)$$

For random $m' \in [0, (a-b) \cdot 2^k[$ the probability density of the cumulative sum can therefore be approximated by:

$$f(x) = \sum_{m'=0}^{(a-b)\cdot 2^k - 1} \frac{1}{(a-b)2^k \sigma_{m'} \sqrt{2\pi}} \exp\left(-\frac{(x - \mu_{m'})^2}{2\sigma_{m'}^2} \right)$$

where $\sigma_{m'} = \sqrt{V_{m'}}$. We have:

$$\sigma_{m'} > \sigma = \sqrt{N} \cdot \sqrt{\frac{(b+1)^2 - 1}{12}}$$

where σ is the same as for the original Floating Mean method.

As previously we obtain a multimodal distribution. The distance between the means of successive components is $\mu_{m'+1} - \mu_{m'} = N \cdot 2^{-k}$ and the standard deviation of a component is at least σ. Therefore the cogs become indistinguishable when $\sigma \gg N \cdot 2^{-k}$ which gives the condition:

$$b \gg \sqrt{N} \cdot 2^{-k}$$

instead of $b \gg \sqrt{N}$ for the original Floating Mean. Therefore by selecting a sufficiently large k we can accommodate a large number of short random delays (large N and small b). In practice, already for k as small as 3 the effect is considerable; we confirm this practically in Sect. 5.3.

We now proceed to compute the mean and variance of the cumulative sum for random m'. Let denote by S_N the sum of the N delays. We have from (2):

$$\mathrm{E}[S_N] = \mathrm{E}[\mu_{m'}] = N \cdot \left(\frac{a}{2} - 2^{-k-1} \right)$$

which is the same as the original Floating Mean up to the 2^{-k-1} term.

To compute the standard deviation of S_N, we represent an individual delay as a random variable $d_i = m + v_i$ where $m = \lfloor m' \cdot 2^{-k} \rfloor$ and v_i is a random variable in the interval $[0, b+1]$. Since m' is uniformly distributed in $[0, (a-b) \cdot 2^k[$, the integer m is uniformly distributed in $[0, a-b[$; moreover the distribution of v_i is independent of m and the v_i's are identically distributed. From

$$S_N = \sum_{i=1}^{N} d_i = Nm + \sum_{i=1}^{N} v_i \, .$$

we get:

$$\mathrm{Var}(S_N) = N^2 \cdot \mathrm{Var}(m) + N \cdot \mathrm{Var}(v_1)$$

For large N we can neglect the term $N \cdot \mathrm{Var}(v_1)$ which gives:

$$\mathrm{Var}(S_N) \simeq N^2 \cdot \mathrm{Var}(m) = N^2 \cdot \frac{(a-b)^2 - 1}{12}$$

which is approximately the same as for the original Floating Mean method. As for the original Floating Mean the variance of the sum of N delays is in $\Theta\left(N^2\right)$ in comparison to plain uniform delays and the method of [8] that both have variances in $\Theta\left(N\right)$.

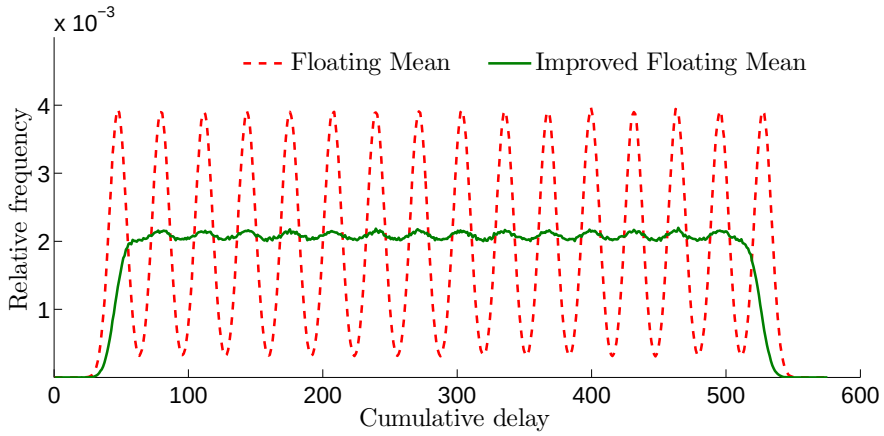


Fig. 3. Improved Floating Mean with $k = 3$ compared to the original method of [3]; $a = 18$, $b = 3$ and $N = 32$ for both methods.

4.2 Illustration

The result is shown in Figure 3 compared to the original Floating Mean. The parameters of both methods were the same as in Figure 2: $a = 18$, $b = 3$ and

$N = 32$. We take $k = 3$ for our Improved Floating Mean method. We can see that we have flattened out the cogs while keeping the same mean. This is because we still have $\sigma \simeq 6.3$ but the distance between successive cogs is now $N \cdot 2^{-k} = 32 \cdot 2^{-3} = 4$ instead of 32 so the cogs are now almost indistinguishable.

4.3 Full Algorithm

The Improved Floating Mean method is formally defined by Algorithm 1.1. Following [3], by $\mathcal{DU}[y, z[$ we denote discrete uniform distribution on $[y, z[$, $y, z \in \mathbb{Z}$, $y < z$. Note that as in [3] we apply the technique of "flipping" the mean in the middle of the execution to make the duration of the entire execution independent of m'. In Appendix B we show that Improved Floating Mean can be efficiently implemented on a constrained platform by describing an implementation in assembly language.

Algorithm 1.1 Improved Floating Mean

Input: $a, b, k, M \in \mathbb{N}, b \leq a, N = 2M$
  $m' \leftarrow \mathcal{DU}[0, (a-b) \cdot 2^k[$
  for $i = 1$ to $N/2$ **do**
    $d_i \leftarrow \lfloor (m' + \mathcal{DU}\left[0, (b+1) \cdot 2^k\right[) \cdot 2^{-k} \rfloor$
  end for
  for $i = N/2 + 1$ to N **do**
    $d_i \leftarrow \lfloor (a \cdot 2^k - m' - \mathcal{DU}\left[0, (b+1) \cdot 2^k\right[) \cdot 2^{-k} \rfloor$
  end for
Output: $d_1, d_2, \ldots, d_N$

5 The Optimal Criterion of Efficiency

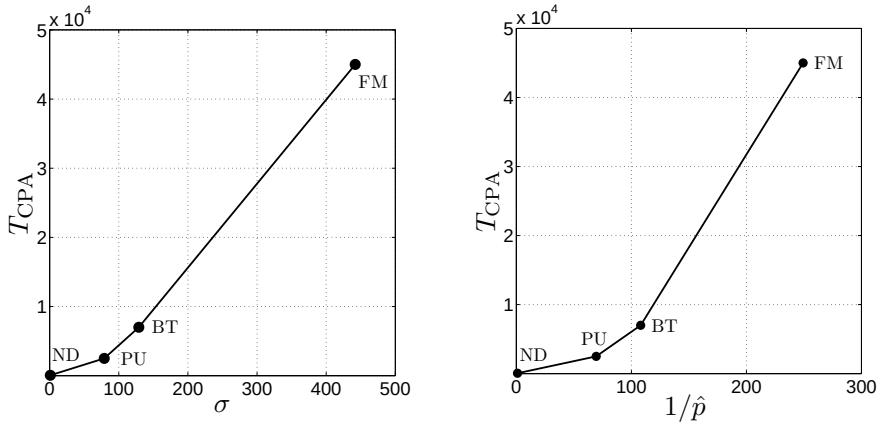
In [3], the ratio σ/μ called coefficient of variation was suggested as a criterion for measuring the efficiency of the random delays countermeasure, where σ is the standard deviation of the cumulative sum of the delays, and μ the mean of the cumulative sum[2], where a higher ratio meant a better efficiency. Here we argue that this measure is misleading and suggest a new criterion.

5.1 Drawbacks of the Coefficient of Variation

We first take a closer look at the experimental data from [3] and establish consistency with the theoretical expectations. In Figure 4(a) we present the relation between the standard deviation σ of the cumulative sum of 32 delays and the

[2] Here σ is the standard deviation of the cumulative sum across various executions, as opposed to Sections 2 and 4 where σ was the standard deviation for a single execution with a fixed m

number T_{CPA} of traces required for a successful CPA attack on the implementation without the random delays countermeasure (no delays, ND) and with different random delay generation methods: plain uniform (PU), Benoit-Tunstall (BT), Floating Mean (FM). The data were taken from Table 2 of [3].



(a) T_{CPA} against σ (b) T_{CPA} against $1/\hat{p}$

Fig. 4. Attack complexity as a function of cumulative sum distribution parameters

Following [2] and [5] one would expect that without integration, which was the case for the experiments in [3], the number of traces grows quadratically with the standard deviation σ. However, from Figure 4(a) one can see that T_{CPA} exhibits growth with σ that is almost linear and not quadratic as one would have expected.

The problem is that standard deviation σ is in general a very rough way to estimate the number of traces which only works for very similar distributions (like two normal distributions). If we look at the figures in Table 2 in [3], we will see that σ for Floating Mean is 5.3 times larger than that for the plain uniform delays. If one expects the number of traces to be in σ^2 in the attack without integration, then $5.3^2 = 28$ more traces are expected to attack the Floating Mean. But observed was only $45000/2500 = 18$ times increase. This means that by looking at the variance, one can overestimate the security level of the countermeasure.

We illustrate this with a simple example. Consider the uniform distribution U of integers on some interval $[a, b]$, $a, b \in \mathbb{Z}$, and the distribution X with $\Pr[X = a] = \Pr[X = b] = 1/2$. We have $\text{Var}(U) = ((a - b + 1)^2 - 1)/12$ and $\text{Var}(X) = (a-b)^2/4$, so $\text{Var}(X) > \text{Var}(U)$. Therefore the efficiency of X counted in σ/μ will be higher than for U. But with X the DPA signal is only divided by 2 instead of $(b-a+1)$ with U, so the number of traces required to break an implementation

with U will be smaller than with X. So in this case the criterion from [3] is misleading.

An accurate estimate is the maximum $\hat{p}$ of the probability mass function (p.m.f.)[3] of the distribution of the cumulative sum. From [2] and [5] we recall that the number of traces T required for a DPA attack is determined by the maximal correlation coefficient ρ_{max} observed in the correlation trace for the correct key guess. Namely, the number of traces can be estimated as

$$T = 3 + 8 \left(\frac{Z_\alpha}{\ln\left(\frac{1+\rho_{max}}{1-\rho_{max}}\right)} \right)^2 \tag{3}$$

where Z_α is a quantile of a normal distribution for the 2-sided confidence interval with error $1-\alpha$. For $\rho_{max} < 0.2$, $\ln\left(\frac{1+x}{1-x}\right) \approx 2x$ holds, so we can approximate (3) for $Z_{\alpha=0.9} = 1.282$ as $T \approx 3/\rho_{max}^2$. So the number of traces is in ρ_{max}^{-2}. In turn, the effect of the timing disarrangement on ρ_{max} is in $\hat{p}$ in case no integration is used. So the number of traces is in $1/\hat{p}^2$.

We now compute $\hat{p}$ for the distributions shown in Figure 2 and plot T_{CPA} from [3] against $1/\hat{p}$ in Figure 4(b). We can now see that quadratic dependency has become clear, which can be verified by the computations: $\hat{p}$ suggests that the number of traces for the Floating Mean will be 13 times higher than for plain uniform delays. Now we have underestimation, but the relation of the number of traces to $\hat{p}$ is still more accurate than to σ. Note that such calculations will not hold for the case without delays since in this case ρ_{max} was about 0.6 (see Figure 5 in [3]), whereas in the other cases $\rho_{max} < 0.2$ holds.

5.2 The New Criterion

We propose a better criterion for the efficiency E of the random delays countermeasure:

$$E = 1/(2\hat{p} \cdot \mu)$$

where $\hat{p}$ is the maximum of the probability mass function and μ is the mean of the distribution of the cumulative sum of the delays. This criterion is normalized and *optimal* with respect to the desired properties of the countermeasure, as shown below.

With the countermeasure, we want to maximize the number of traces in an attack, *i.e.* minimize $\hat{p}$, while keeping the smallest possible overhead, *i.e.* smallest mean μ. One can see that from all distributions with the given $\hat{p}$, the one with the smallest μ is uniform on $[0, 1/\hat{p}]$. In this case, $\mu = 1/(2\hat{p})$ and the criterion E is equal to 1. In all other cases (same $\hat{p}$ but larger μ) the value of the criterion will be smaller, and the closer to zero – the farther is the distribution from the optimal one (*i.e.* the uniform one).

[3] and not p.d.f. since the distribution is discrete

This tightly relates the criterion to the entropy of the distribution. Namely, the new criterion is directly linked to min-entropy, which is defined for a random variable S as

$$H_\infty(S) = -\log \max_i p_i = -\log \hat{p}.$$

Note that $H_\infty(S) \leq H(S)$, where $H(S) = -\sum_i p_i \log p_i$ is the Shannon entropy, so min-entropy can be considered as a worst-case measure of uncertainty. Now we have $\hat{p} = 2^{-H_\infty(S)}$ and the new efficiency criterion is expressed as

$$E = \frac{2^{H_\infty(S)-1}}{\mu}.$$

Indeed, for a fixed worst-case cumulative delay, the distributions with the higher entropy, *i.e.* maximizing uncertainty for the attacker, will have lower $\hat{p}$, larger number of traces to attack and thus more efficient as a countermeasure.

This criterion is easily computable once the designer have simulated the real distribution for the concrete parameters of a method (taking into consideration the number of clock cycles per delay loop) and obtained $\hat{p}$.

5.3 Comparing Efficiency

In our example in Sect. 4.2 with parameters $a = 18$, $b = 3$ and $N = 32$, with the Improved Floating Mean (IFM) method we have decreased $\hat{p}$ by a factor 2, as illustrated in Figure 3. So the number of traces for the successful straightforward DPA attack will be in principle almost 4 times larger (around 160000), and according to the optimal criterion the efficiency is almost 2 times higher. Table 1 below revises Table 2 of [3] with the new criterion and the new method.

Table 1. New efficiency criterion for different methods

	ND	PU	BT [8]	FM [3]	IFM [this paper]
μ, cycles	0	720	860	862	953
$\hat{p}$	1	0.0144	0.0092	0.0040	0.0020
$E = 1/(2\hat{p}\mu)$	–	0.048	0.063	0.145	0.259
T_{CPA}, traces	50	2500	7000	45000	> 150000

We have performed a practical power analysis attack against an AES-128 implementation with the new IFM method running on ATmega16, an 8-bit AVR microcontroller. To be consistent with the previous results, the implementation and the measurement setup were as in [3]. Namely, there were 10 random delays per round, and 3 dummy rounds were added before and after the encryption, so $N = 32$ delays occur between the start of the execution (the synchronization point) and the 1-st S-Box lookup of the 1st encryption round the attack target.

The parameters for IFM were $a = 19$, $b = 3$, $k = 3$. Note that a different value for a was chosen to ensure efficient implementation of the method as described in Appendix B, so the mean for IFM is larger, but still the efficiency is 1.8 times higher. We could not break this implementation by a CPA attack [1] with $150 \cdot 10^3$ traces, which corresponds to the theoretical expectations.

Due to its definition, the new criterion reflects well the number of traces observed in the experimental attack. For example, looking at the new criterion we expect the number of traces for the Floating Mean be $0.145^2/0.063^2 = 5.3$ times higher than for the table method of Benoit and Tunstall [8]. In the experiment, it was $45000/7000 = 6.4$ times higher.

6 Conclusion

We have shown that the Floating Mean method for random delay generation in embedded software [3] exhibits lower security if its parameters are improperly chosen. We have suggested how to choose the parameters of the method so that it generates a good distribution; however this requires to generate longer delays while in practice it is preferable to have multiple shorter delays. We have proposed an improved method that allows for a wider choice of parameters while having an efficient implementation. Finally, we have suggested an optimal criterion for measuring the efficiency of the random delays countermeasure.

References

1. E. Brier, C. Clavier, and O. Benoit. Correlation power analysis with a leakage model. In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 135–152. Springer, Heidelberg, 2004.
2. 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, *CHES 2000*, volume 1965 of *LNCS*, pages 252–263. Springer, Heidelberg, 2000.
3. J.-S. Coron and I. Kizhvatov. An efficient method for random delay generation in embedded software. In C. Clavier and K. Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 156–170. Springer, Heidelberg, 2009.
4. N. Homma, S. Nagashima, T. Sugawara, T. Aoki, and A. Satoh. A high-resolution phase-based waveform matching and its application to side-channel attacks. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E91-A(1):193–202, 2008.
5. S. Mangard. Hardware countermeasures against DPA – a statistical analysis of their effectiveness. In T. Okamoto, editor, *CT-RSA 2004*, volume 2964 of *LNCS*, pages 222–235. Springer, Heidelberg, 2004.
6. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.
7. G. McLachlan and D. Peel. *Finite Mixture Models*. John Wiley & Sons, 2000.
8. M. Tunstall and O. Benoit. Efficient use of random delays in embedded software. In D. Sauveron, K. Markantonakis, A. Bilas, and J.-J. Quisquater, editors, *WISTP 2007*, volume 4462 of *LNCS*, pages 27–38. Springer, Heidelberg, 2007.
9. J. G. J. van Woudenberg, M. F. Witteman, and B. Bakker. Improving Differential Power Analysis by elastic alignment. http://www.riscure.com/fileadmin/images/Docs/elastic_paper.pdf, 2009.

A Distribution of Delay's Length d

We have:

$$\mathrm{E}[d] = \frac{1}{(b+1)2^k} \sum_{i=m'}^{m'+(b+1)\cdot 2^k-1} \lfloor i \cdot 2^{-k} \rfloor$$

Write $m' = m \cdot 2^k + u$ with $0 \le u < 2^k$. This gives:

$$\begin{aligned}
\mathrm{E}[d] &= \frac{1}{(b+1)2^k} \sum_{i=m2^k+u}^{(m+b+1)2^k+u-1} \lfloor i \cdot 2^{-k} \rfloor \\
&= \frac{1}{(b+1)2^k} \left(\sum_{i=m2^k+u}^{(m+1)2^k-1} \lfloor i \cdot 2^{-k} \rfloor + \sum_{i=(m+1)2^k}^{(m+b+1)2^k-1} \lfloor i \cdot 2^{-k} \rfloor + \sum_{i=(m+b+1)2^k}^{(m+b+1)2^k+u-1} \lfloor i \cdot 2^{-k} \rfloor \right) \\
&= \frac{1}{(b+1)2^k} \left(m \cdot (2^k - u) + 2^k \sum_{j=m+1}^{m+b} j + (m+b+1) \cdot u \right) \\
&= \frac{1}{(b+1)2^k} \left(m \cdot 2^k + (b+1) \cdot u + b \cdot 2^k \cdot \left(m + \frac{b+1}{2} \right) \right) \\
&= \frac{1}{(b+1)2^k} \left(m \cdot (b+1) \cdot 2^k + (b+1) \cdot u + b \cdot 2^k \cdot \frac{b+1}{2} \right) \\
&= m + u \cdot 2^{-k} + \frac{b}{2} = m' \cdot 2^{-k} + \frac{b}{2}
\end{aligned}$$

Similarly we have:

$$\begin{aligned}
\mathrm{E}[d^2] &= \frac{1}{(b+1)2^k} \sum_{i=m2^k+u}^{(m+b+1)2^k+u-1} \lfloor i \cdot 2^{-k} \rfloor^2 \\
&= \frac{1}{(b+1)2^k} \left(\sum_{i=m2^k+u}^{(m+1)2^k-1} \lfloor i \cdot 2^{-k} \rfloor^2 + \sum_{i=(m+1)2^k}^{(m+b+1)2^k-1} \lfloor i \cdot 2^{-k} \rfloor^2 + \sum_{i=(m+b+1)2^k}^{(m+b+1)2^k+u-1} \lfloor i \cdot 2^{-k} \rfloor^2 \right) \\
&= \frac{1}{(b+1)2^k} \left(m^2 \cdot (2^k - u) + 2^k \sum_{j=m+1}^{m+b} j^2 + (m+b+1)^2 \cdot u \right)
\end{aligned}$$

After simplifications this gives:

$$\mathrm{Var}[d] = \mathrm{E}[d^2] - \mathrm{E}[d]^2 = \frac{(b+1)^2 - 1}{12} + 2^{-k} \cdot u(1 - 2^{-k}u)$$

B Efficient Implementation of Improved Floating Mean

Here we show that our new Improved Floating Mean method can be efficiently implemented and introduces only a slight additional performance overhead compared to the original Floating Mean (*cf.* Appendix B of [3]).

In the Improved Floating Mean method one has to generate the mean and the individual delays in a broader range but then round them. The former is done by modifying the mask for truncating the random numbers so it is k bits longer, the latter – by shifting the register with the delay right by k bits.

As a reference, the new implementation of the delay loop in the 8-bit AVR assembly (*cf.* [3]) for the Improved Floating Mean is:

```
    rcall randombyte    ; obtain a random byte in RND
    and   RND, MASKBK   ; truncate to the desired length including k
    add   RND, FM       ; add 'floating mean'
    lsr   RND           ;
    ...                 ; logical shit right by k bits
    lsr   RND           ;
    tst   RND           ; balancing between zero and
    breq  zero          ;  non-zero delay values
    nop
    nop
 dummyloop:
    dec   RND
    brne  dummyloop
 zero:
    ret
```

and the generation of m in register `FM` in the beginning of the execution looks like:

```
    rcall randombyte    ; obtain a random byte in RND
    and RND, MASKMK     ; truncate to the desired length including k
    mov FM, RND         ; store 'floating mean' on register FM
```

Here, the masks have the following form:

$$\texttt{MASKBK} = 0\ldots0\underbrace{1\ldots1}_{t}\underbrace{1\ldots1}_{k}$$
$$\texttt{MASKMK} = 0\ldots0\underbrace{1\ldots1}_{s}\underbrace{1\ldots1}_{k}$$

where $2^t - 1 = b$ and $2^s = a - b$ (we note that this choice of parameters is slightly different from the one for efficient implementation of the Floating Mean, and therefore a was set to 19 for $b = 3$ in our experiments reported in Sect 5.3). To ensure that the operations are performed on a single register and no overflow occurs on an n-bit microcontroller, s, t, and k should be chosen such that $\max(s, t) + k + 1 \leq n$.

Note that the number of cycles per delay loop itself did not change. What changed is the additional overhead per delay. In the case of the 8-bit AVR implementation it is k additional cycles required for k-bit shift right. For a small k like $k = 3$ the impact is therefore insignificant.