Side-Channel Attack Countermeasures Based On Clock Randomization Have a Fundamental Flaw

Martin Brisfors* Michail Moraitis* Elena Dubrova

Department of Electrical Engineering,
Royal Institute of Technology (KTH)
Electrum 229, 196 40 Stockholm, Sweden
{brisfors, micmor, dubrova}@kth.se

Abstract

Clock randomization is one of the oldest countermeasures against side-channel attacks. Various implementations have been presented in the past, along with positive security evaluations. However, in this paper we show that it is possible to break countermeasures based on a randomized clock by sampling side-channel measurements at a frequency much higher than the encryption clock, synchronizing the traces with pre-processing, and targeting the beginning of the encryption. We demonstrate a deep learning-based side-channel attack on a protected FPGA implementation of AES which can recover a subkey from less than 500 power traces. In contrast to previous attacks on FPGA implementations of AES which targeted the last round, the presented attack uses the first round as the attack point. Any randomized clock countermeasure is significantly weakened by an attack on the first round because the effect of randomness accumulated over multiple encryption rounds is lost.

keywords: Side-channel attack, Random Execution Time , Randomized Clock , Countermeasure, Oversampling, Deep Learning, FPGA, AES, Correlation Power Analysis.

1 Introduction

The idea of randomizing the execution time of cryptographic algorithms to protect implementations against side-channel attacks is as old as the attacks themselves [KJJ99, KJJ]. There have been numerous papers exploring the topic in the past, mostly for software implementations [KJJR11]. Several papers have

*Both authors contributed equally to this manuscript.

also suggested ways of implementing randomized clocks to protect FPGA implementations. Their security evaluations have shown that the proposed countermeasures are resistant to Differential Power Analysis (DPA)/Correlation Power Analysis (CPA) [BLGT05, LOM08, ZH08, ZPH10, BHL+10, BLOW10, GM11, RBBC18, CK09, CK10, JIP19, HDL+20], or Deep Learning (DL)-based EM analysis [HDL+20].

However, in this paper we show that it is possible to break countermeasures based on clock randomization by using a sampling frequency that is much higher than the clock of the cryptographic implementation, synchronizing the traces with pre-processing, and carefully selecting the attack point. The main contributions of the paper are:

- We highlight the importance of oversampling in side-channel analysis by demonstrating how a seemingly secure countermeasure can be broken when the oversampling rate is sufficiently high. Ignoring oversampling may lead to an overestimation of security of randomized clock countermeasures.
- We present a deep learning-based side-channel attack on a protected FPGA implementation of AES which targets the first round. To the best of our knowledge, all previous attacks on an FPGA implementation of AES with randomized clock targeted the last round [BLGT05, LOM08, ZH08, ZPH10, BHL+10, BLOW10, GM11, RBBC18, CK09, CK10, JIP19, HDL+20]. Attacking the first round significantly weakens any randomized clock countermeasure because the effect of randomness accumulated over the multiple encryption rounds is lost.
- We propose a randomized clock implementation in which the varying frequency is achieved by an asynchronous switching between four stable frequencies using a fifth frequency and a random number generator.

The rest of the paper is organized as follows. Section 2 reviews the previous work on randomized clock countermeasures. Section 3 presents our randomized clock implementation. Section 4 describes the methods which we use to break the randomized clock countermeasure. Section 5 summarizes the experimental results. Finally, Section 6 concludes the paper.

2 Previous work

Various techniques for randomizing the execution time of cryptographic algorithms have been proposed over the years to protect implementations against side-channel attacks, including using randomized clocks, the addition of random delays or dummy operations, random execution re-ordering, and random branching [KJJR11]. In hardware, the majority of countermeasures focus on inserting random delays, or randomizing the clock.

To the best of our knowledge, the first hardware-based random delay insertion method was presented in [BLGT05]. The main idea is to insert random

delays in the datapath of a cryptographic processor in order to randomize its power consumption profile. To realize that, a gate-level implementation of a random delay and a random wait flip-flop were proposed. In [LOM08] the concept is transferred to FPGAs and the resistance of the resulting countermeasure to Differential Power Analysis (DPA) is evaluated.

In [BHL+10] a key scheduler controlled by a True Random Number Generator (TRNG) is introduced to create the effect of a randomized clock. To achieve that, the scheduler randomly selects between the output of a set of positive edge and negative edge flip-flops. The approach is further extended in [BLOW10] by injecting dummy data during the idle periods.

In [ZH08] a design that uses a different frequency to encrypt each 128-bit plaintext block of an AES-128-CTR FPGA implementation is presented (CTR stands for the counter mode operation of a block cipher). In this design, multiple frequencies $f_1, \ldots, f_n$ are generated by dividing a base frequency f given by a single-input ring oscillator. In addition, four out of the n generated frequencies are phase shifted using Digital Clock Managers (DCMs) to obtain their phase shifted copies $\{f_{i\_0^\circ}, f_{i\_90^\circ}, f_{i\_180^\circ}, f_{i\_270^\circ}\}$, for $i \in \{1, 2, 3, 4\}$. The selection among multiple frequencies is performed pseudo-randomly using a multiplexer controlled by a Linear Feedback Shift Register (LFSR). Later, in [ZPH10] it is suggested that performing frequency switching in every clock cycle instead of every new plaintext block offers greater security. In [GM11], following the same idea, a base clock frequency f is phase shifted through DCMs to create n new frequencies, each shifted by $360/n$ degrees. The selection among multiple frequencies is performed using a clock multiplexer tree controlled by a TRNG. In [RBBC18], the design presented in [GM11] is improved by incorporating the floating mean method [CK09, CK10] to generate uniform random numbers.

In [JIP19] an implementation that leverages the ability of Mixed-Mode Clock Management (MMCMs) to be reconfigured at runtime is presented. The FPGA Block RAMs (BRAMs) are utilized to store different MMCM configurations that define selected sets of frequencies. For each MMCM, m possible configurations, with n output frequencies defined in each, are stored. The implementation proposed in [JIP19] uses $n = 3$ and $m = 1024$, resulting in $3,072$ different clock frequencies in total. Since the reconfiguration of MMCMs takes a considerable amount of time (equivalent to 82 encryptions in their case), at least two MMCMs have to be deployed so that one is working while the other one is being reconfigured to achieve runtime frequency tuning. As in [GM11], the selection among frequencies is performed using a multiplexer controlled by an RNG. The authors use the number of different *cumulative completion times* generated by their approach as a metric of its resilience. With an AES-128 implementation that takes $r = 10$ clock cycles to complete an encryption, the number of different cumulative times to completion is calculated as ${}_{r+n-1}C_r \times m = 66 \times 1024 = 67,584$.

In [HDL+20] a more lightweight and scalable solution that can generate even more different times to completion is presented. To generate different frequencies, MMCMs apply a scaling factor to an input clock. Thus, in a constant MMCM configuration, different input frequencies produce different output frequencies. Taking that into consideration, the authors of [HDL+20] propose a

design that consists of a software-based clock randomizer that generates frequencies in a given range that are fed into an MMCM with constant configuration. The clock randomizer module consists of a processing system core (available in modern FPGAs such as Xilinx Ultrascale) that controls the configuration of two programmable clock dividers. A Phase-Locked Loop (PLL) creates a stable clock frequency that passes through the clock dividers and is subsequently fed into an MMCM through a glitch-free clock gate. The MMCM generates n different frequencies by applying the preset scaling factors to the input frequency. Finally, the selection among these n frequencies is done using an RNG-controlled clock multiplexer (that allows glitch-free switching between clocks) which outputs the randomized clock used for the encryption.

The advantage of this approach over the one in [JIP19] is that it does not require the use of BRAMs to store different MMCM configurations[1] and that it is more agile since the input frequencies are controlled in software. The authors adopted the *number of cumulative completion times* metric to evaluate their approach. They tested two implementations, one with $n = 4$ MMCM output frequencies and one with $n = 8$. Their AES-128 implementation takes $r = 13$ clock cycles to complete an encryption and uses $m = 257$ different base frequencies. This results in ${}_{r+n-1}C_r \times m = 560 \times 257 = 143,920$ for $n = 4$ and $77,520 \times 257 = 19,922,640$ different cumulative times to completion for $n = 8$.

In [JIP21], a lightweight countermeasure offering resistance to remote power attacks for up to one million encryptions is presented. The idea is to add noise to the timing measurements of digital converters which are used in the remote attacks. This is realized by adding a random delay in the magnitude of picoseconds to each clock cycle.

3 Proposed randomized clock implementation

The block diagram of the proposed randomized clock is shown in Fig. 1. It consists of an MMCM block, five global clock simple buffers (BUFG), two FDRE D flip flops, a 4-to-1 multiplexer and a 2 input AND gate. A detailed description of the Xilinx 7 series FPGA components referenced in this work is available in the user guide UG953 [Xil22].

The MMCM block has one clock input, four clock outputs and a locked signal to indicate when the generated clocks are ready to be used. The generated clocks pass through one BUFG each and then connect to the 4-to-1 multiplexer. In the implementations presented in the related work [JIP19, HDL+20], the multiplexers are implemented through a tree of 2-to-1 global clock MUX buffers (BUFGMUX_CTRL). A BUFGMUX_CTRL is a primitive of Xilinx 7 series FPGAs that allows a clean, glitch-free switching between two input frequencies. To achieve that, whenever setup/hold conditions are about to be violated by the switching of the select signal, the output clock appears one clock later. In our implementation, we do not want to have this functionality because our aim

[1] An alternative design with dynamically reconfigurable MMCMs is mentioned, but not implemented.

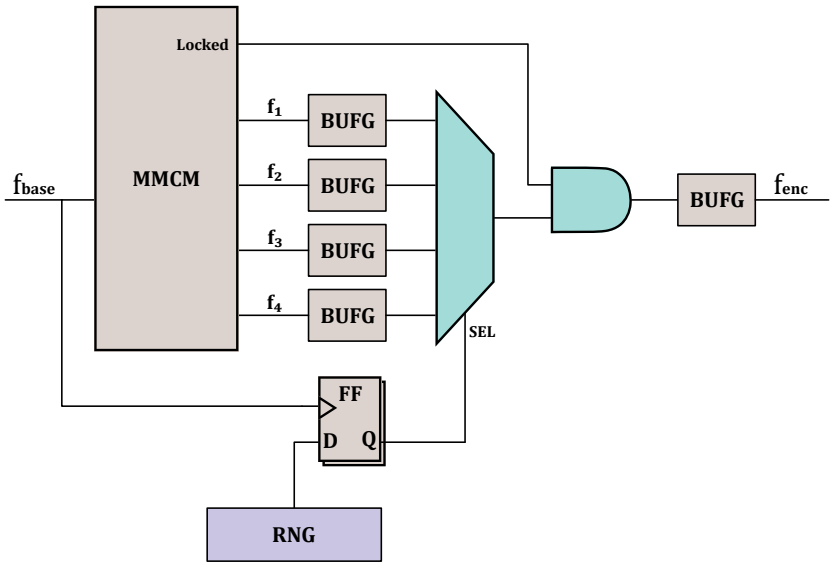

Figure 1: The block diagram of the proposed randomized clock implementation.

is to have the selector signal forcefully change between frequencies to create a randomized clock. Such a design choice leads to a clock whose cycles are occasionally too short for the AES core to complete the encryption in time. Therefore, the multiplexer in our design can be implemented either by three BUFGMUXs set to asynchronous switching,[2] or a single 6-input 2-output look up table (LUT6).

The selector signal comes from two FDRE D flip flops that are clocked with the base frequency and store values from a random number generator (RNG). The RNG can be a TRNG or a PRNG (e.g. an LFSR). In our experiments we used a set sequence of pre-generated pseudo-random numbers. By adding these two registers, the random numbers arrive at the multiplexer with half the frequency of the base clock, regardless of the frequency of the RNG implementation. This assures a stable select signal.

Finally, the randomized frequency created at the output of the multiplexer is combined with the locked signal of the MMCM (high when the MMCM frequencies are ready to be used) using a 2-input AND gate. Its output passes through a BUFG which outputs the randomized clock used for encryption.

In Fig. 2 the randomized clock generation of the presented implementation is illustrated. On every positive edge of the base clock, f_{base}, the randomized clock's value, f_{enc}, switches asynchronously to the value of one of the four clocks, f_1, f_2, f_3, f_4, depending on the value of the select signal, SEL, generated by

[2] A BUFGMUX_CTRL is a Global Clock Control Buffer (BUFGCTRL) with the clock enable (CE) inputs set to constant 1 and select inputs (S) connected to the selection signal. When the select signal is connected to the CE inputs, the glitch-free functionality is lost and the switching occurs asynchronously.

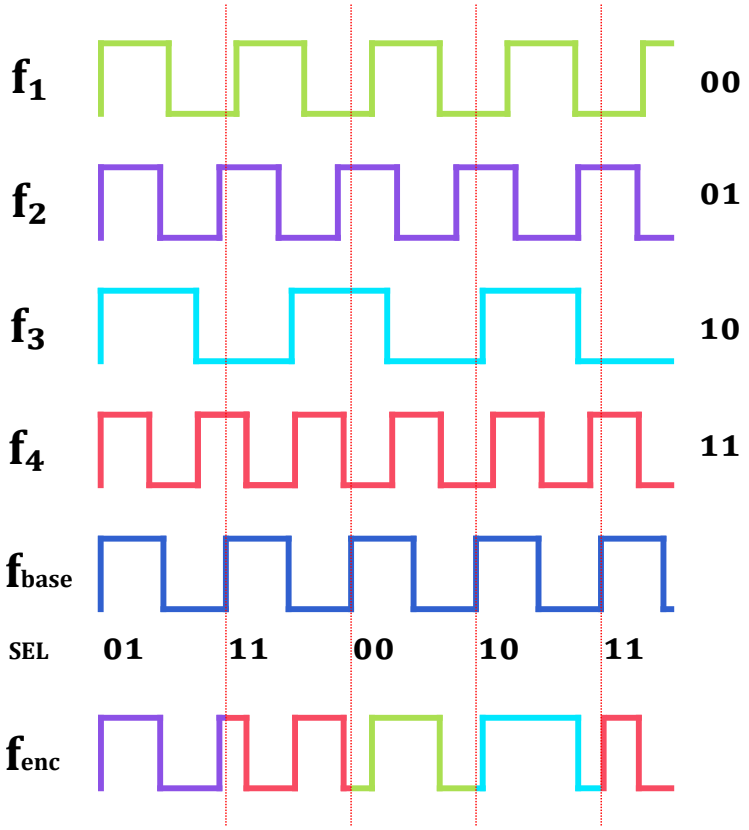

Figure 2: Proposed randomized clock generation.

the TRNG. To assess how many different frequencies our implementation can generate, we performed a simulation. In the simulation, each base clock cycle is represented with a precision of 100K samples and the clocks are simulated 3.7M times. The results show that our randomized clock implementation can generate pulses of at least 403 different frequencies. Therefore, assuming an AES implementation with $r = 10$, the number of cumulative completion times of our randomized clock is ${}_{r+n-1}C_r \times m = {}_{10+403-1}C_{10} \times 1 \approx 3.478 \times 10^{19}$.

4 Side-channel analysis in the presence of a randomized clock

In this section we describe the methods we use to break the randomized clock countermeasure: oversampling, trace pre-processing, and CPA and DL-based side-channel analysis.

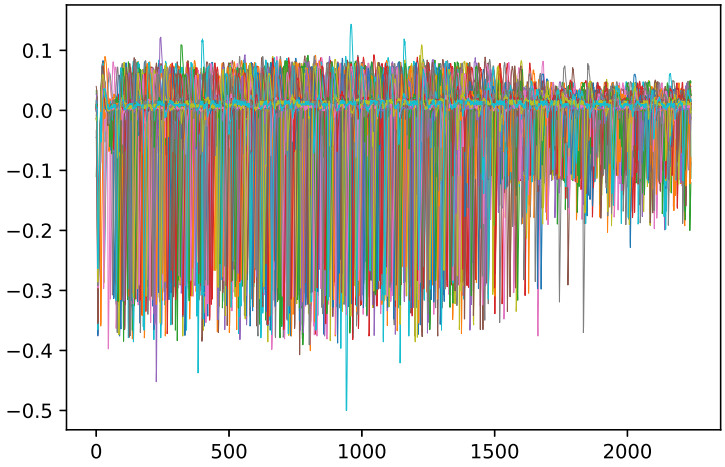

Figure 3: 100 randomly selected power traces captured with 40× oversampling representing a complete encryption.

4.1 Oversampling

In signal processing, *oversampling* is the process of sampling a signal at a sampling frequency significantly higher than the Nyquist rate[3]. A signal is said to be oversampled by a factor of N if it is sampled at N times the Nyquist rate. It is known that oversampling can improve resolution and signal-to-noise ratio [Sch00].

4.2 Pre-processing

Several pre-processing methods have been proposed to combat trace misalignment and re-enable a successful analysis. These methods include dynamic time wrapping (DTW)/elastic alignment [vWWB11], pattern matching [ACPJ17], fast Fourier transform (FFT) [SDB+10], principal component analysis (PCA) [HB10], rapid alignment method (RAM) [MvWB11] and sliding window (SW) [FW18].

In this paper, we test two pre-processing methods: (1) sliding window [FW18] with a window size of 20 and (2) trace synchronisation based on deviation from mean. We apply trace synchronisation on sufficiently oversampled power measurements using the following simple approach.

By setting a threshold based on the deviation from the mean of the value being more than 1.5σ, we identify peaks corresponding to the rising edge of the randomized clock which is used for encryption. As one can see from Fig. 5, traces captured with a high oversampling factor have distinct power peaks. These peaks can be used to synchronize traces for a specific round of the algorithm. The lower the oversampling factor is, the less distinct the peaks are. So, traces

[3]The Nyquist rate is defined as twice the bandwidth of the signal.

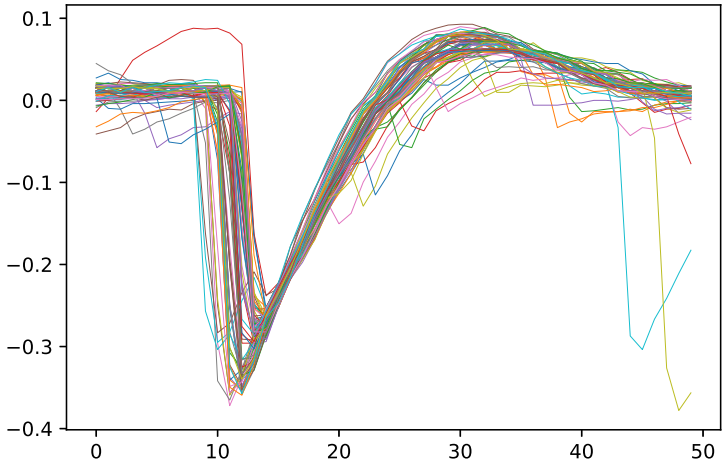

Figure 4: A zoomed-in interval of traces from Fig. 3 representing the first round of AES after synchronization. This interval is given as input to MLPs.

captured with a high oversampling factor are easier to synchronize. They are also more likely to accurately reflect the power consumption.

Using the threshold, we synchronize traces for the first round of AES by finding, for each trace, the first point where the trace crosses the threshold after setup instructions are completed. Selecting a window around this point yields a good synchronization as one can see in Fig. 4.

A similar synchronization strategy can be applied to the last round of AES by going backwards.

4.3 Correlation Power Analysis

We perform the CPA in a usual way [BCO04], by assigning a power hypothesis to every trace in the data set, for every subkey guess, with the subkey size being a byte. The Hamming distance between the states of round 9 and round 10 is used as a power hypothesis:

$$P_i = HW(ShiftRows^{-1}(c_i) \oplus Sbox^{-1}(c_i \oplus RK10_i)),$$

where c_i is the ith byte of the ciphertext, $RK10_i$ is the guess for 10th round subkey i, and HW is the Hamming weight, for $i \in \{1, 2, \ldots, 16\}$.

To recover each subkey, we calculate the Pearson correlation coefficient for the corresponding power hypotheses and trace measurements and choose the subkey guess which maximizes the absolute value of the correlation coefficient.

4.4 Deep Learning-Based Analysis

It is known that deep learning-based side-channel analysis can handle trace misalignment caused by jitter without any pre-processing [RQL19]. This makes

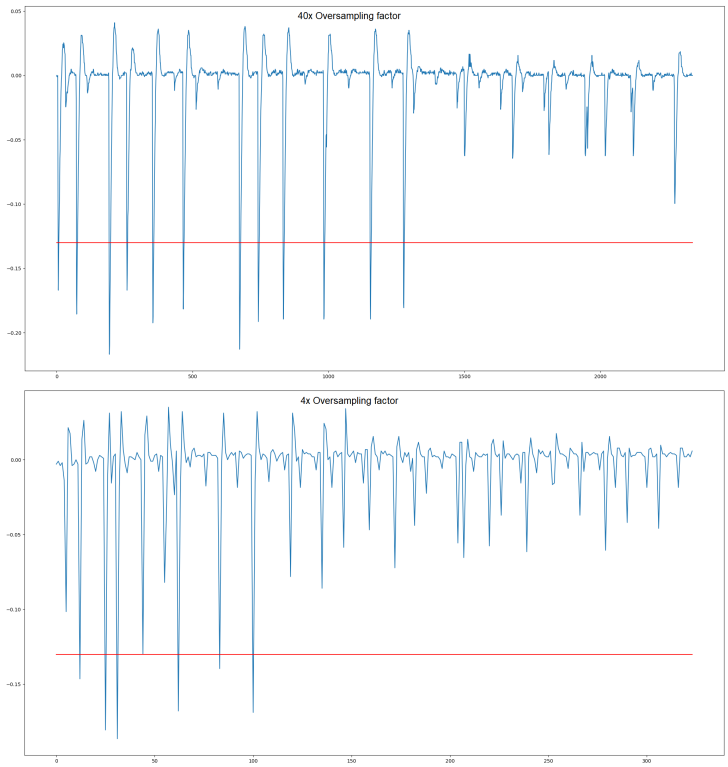

Figure 5: Comparison of traces captured with 40× and 4× oversampling using the same threshold for identifying peaks. Insufficient oversampling may lead to an incorrect sampling/measurement of peak power consumption.

it a good choice for dealing with temporal noise introduced by randomized clocks.

We use the profiling approach in which, at the profiling stage, a neural network model is trained to learn the power profile of the target algorithm implementation for all possible values of the subkey and, at the attack stage, the model is used to classify traces captured from the device under attack.

We train multilayer perceptron (MLP) neural networks of type $\mathcal{N}_i : R^n \rightarrow I^{256}$ for the subkey $i \in \{1, 2, \ldots, 16\}$, where n is the number of data points in traces, R is the set of real numbers, and $I := \{x \in R \mid 0 \leq x \leq 1\}$. The MLP architecture is listed in Table 1. *Sbox* output values in the first round were used as labels for traces.

For training, we use a set of 10M traces captured for random messages. From this set, 70% is used for training and 30% for validation. The training is carried out using *Nadam* optimizer with the learning rate of 0.002. The maximum number of epoch is set to 12 with a batch size of 1024. Only the model with

the best validation accuracy is saved.

To test the models, we use two metrics:

1. Accuracy of subkey prediction from a single trace, and
2. Average number of traces required for the subkey recovery.

Note that the single-trace prediction accuracy metric is not applicable to CPA since single-trace CPA is impossible. The average number of required traces is defined as the average number of traces required to recover the subkey in the majority of tests performed on a randomly permuted test data set.

4.5 Importance of using the first round as the attack point

In previous work, the last round of AES is used as the attack point. This is because for a typical[4] FPGA implementation of AES, DPA/CPA attacks on the first round are not successful.

However, as our experiments show, the DL-based analysis can recover the key from the first round. Despite of the weaker leakage of the first round as compared to the last, such an approach is preferable when a randomized clock is used as a countermeasure for the following reasons. In a first round attack, the cumulative times to completion metric is reduced to the number of different frequencies. This considerably weakens any randomized clock countermeasure. For example, for a first round attack with $r = 1$ in [HDL+20], the number of different times to completion gets reduced from $67,584$ to ${}_{r+n-1}C_r \times m = 3 \times 1024 = 3,072$ and from $19,922,640$ to ${}_{r+n-1}C_r \times m = 8 \times 257 = 2,056$. Furthermore, when attacking the first round, the overall effect of randomness accumulated[5] over multiple encryption rounds is lost.

5 Experimental Results

This section presents the results of our experiments.

5.1 Equipment

The equipment used for our experiments is a CW1173 ChipWhisperer-Lite and a CW305 Artix 7 FPGA target board.

ChipWhisperer [Newa] is a fully open-source, low-cost toolkit for hardware security evaluation. It handles power trace acquisition and communication of target devices with a computer, making side-channel attacks easier to perform. The power measurements are taken over a shunt resistor connected between the

[4] Here by "typical´´ we mean the AES implementation in which one round is computed per clock cycle and the state is stored at the end of the round.

[5] The cumulative effect of randomness can described by a random walk and the variance of a random walk increases with the walk length. Thus, if the timing shifts are randomly distributed, the uncertainty in the first round is provably smaller than the uncertainty in the last.

Table 1: MLP Architecture.

Layer (type)	Output Shape	Param #
batch_normalization_1	(None, 50)	200
dense_1 (Dense)	(None, 1024)	52224
batch_normalization_2	(None, 1024)	4096
relu_1 (ReLU)	(None, 1024)	0
dense_2 (Dense)	(None, 512)	524800
batch_normalization_3	(None, 512)	2048
relu_2 (ReLU)	(None, 512)	0
dense_3 (Dense)	(None, 256)	131328
batch_normalization_4	(None, 256)	1024
relu_3 (ReLU)	(None, 256)	0
dense_4 (Dense)	(None, 256)	65792
softmax_1 (Softmax)	(None, 256)	0
Total params:	781,512	
Trainable params:	777,828	
Non-trainable params:	3,684	

power supply and the target device. ChipWhisperer-Lite employs a synchronous capture method, which greatly improves trace synchronization while also lowering the required sample rate and data storage. ChipWhisperer-Lite has a buffer size of up to 24,400 samples that can be captured at a maximum sampling rate of 105 MS/sec.

The CW305 target board used in our experiments is equipped with an Artix 7 XC7A35T-2FTG256 FPGA. The cryptographic algorithm implementation in which we integrated our randomized clock countermeasure is Google's AES-128 implementation which can be found in the Vault Project repository [Pro]. The AES core module is integrated into the ChipWhisperer interface [Newb]. The design synthesis and bitstream generation is performed with Vivado 2019.1.

5.2 Trace acquisition with oversampling

Side-channel analysis with a high oversampling rate requires the use of measuring equipment that allows high sampling frequencies. Our equipment, ChipWhisperer-Lite, can handle sampling rates of only up to 105 MS/sec. Therefore, the FPGA must run at 5.25 MHz to approximate a 20× oversampling or at 2.6 MHz to approximate a 40× oversampling. This goes against the MMCM specification which requires an input frequency in the range of 10-800 MHz.

To overcome this limitation imposed by our equipment, we use a 10Mhz base frequency and add a clock divider after each frequency with the division parameter $d = 10$. The frequencies generated by the MMCM are: $f_1 = 11.9713$ MHz, $f_2 = 7.7315$ MHz, $f_3 = 9.2778$ MHz, $f_4 = 12.6515$ MHz, with an $f_{mean} = 10.408$ MHz. When oversampling, we assume an FPGA base frequency of 1 Mhz, e.g. for 20× oversampling we sample at 20 Mhz. Since the frequency of

the randomized clock is unknown, not all traces are oversampled by this nominal degree. Furthermore, the asynchronous frequency switching (discussed in Section 3) causes about 3% of ciphertexts to be incorrect. Incorrect encryptions are typically acceptable in applications in which re-encryption is possible, e.g. the encryption of a nonce for challenge-response authentication.

In a real attack, slowing down the clock of an FPGA implementation would require a different approach, especially if countermeasures to prevent it, e.g. the clock manipulation detector [GM11], are present. In our threat scenario, we do not consider such attacks. We assume that, to oversample, the adversary will use an oscilloscope which can capture traces at a much higher rate than the frequency of the clock used in the encryption core of the FPGA.

5.3 Overhead evaluation

We compared our randomized clock implementation with two state-of-the-art architectures that offer the highest level of side-channel resistance: the Runtime Frequency Tuning Countermeasure (RFTC) [JIP19] and the Dynamic Frequency Randomization (DFR) [HDL$^+$20].

Table 2 lists the FPGA resources and timing overhead. One can see that the presented implementation has the smallest hardware and timing overhead.

5.4 Comparison considerations

To compare countermeasures properly, one has to take into account the number of traces required to break the unprotected implementation, as well as the sampling and operating frequency of the implementations. The number of traces for CPA is shown in Table 4. When comparing these numbers it should be taken into account that [HDL$^+$20] uses EM side-channels while [JIP19] and the presented method use power. EM side-channels are usually noisier and typically attacks require an order of magnitude more traces.

Regarding sampling and operating frequencies, in [JIP19], power measurements are collected using an oscilloscope with a 100 Mhz bandwidth and a maximum sampling rate of 1 Gs/s. The randomized clock frequencies are in the range of 12-48 MHz.

In [HDL$^+$20] an oscilloscope with a bandwidth of 500 Mhz and maximum sampling rate of 5 Gs/s is used. The frequencies of the randomized clock are in the range of 17.5-213.3 Mhz for the implementation based on four clocks, and 17.5-426.5 MHz for the one based on eight clocks.

No information about the sampling frequency is given in [JIP19, HDL$^+$20]. We can make a rough estimation as follows. Taking into account the maximum sampling frequency of their equipment, and considering the mean operating frequency as nominal, in [JIP19] up to 41$\times$ oversampling can be achieved while in [HDL$^+$20] up to 43$\times$ and 23$\times$ for the four and eight clock implementations, respectively. Since there is a big gap between their lowest and highest operating frequencies, these numbers are certainly not accurate. However, they are our best estimate.

Table 2: Overhead comparison with the state-of-the-art implementations of randomized clock.

Method	FPGA resources			Timing
	# BRAM	# MMCM	# BUFG	overhead
RFTC [JIP19]	20	2	N/A	1.72×
DFR [HDL$^+$20]	0	1	12-23	1.54-58.9×
Presented	0	1	5	1.27×

Table 3: CPA and MLP subkey recovery results for different oversampling factors (average of 10 tests).

Over-	Last round attack		First round attack	
sampling	CPA-SYNC	SW-CPA	MLP	
factor	# Traces	# Traces	single-trace acc,%	# Traces
Unprotected 1×	400	1000	0.66	112
Protected 4×	>10M	>10M	0.39	>10M
Protected 10×	50k	>10M	0.49	582
Protected 20×	5k	>10M	0.51	319
Protected 40×	3k	5M	0.52	430

In our case, we sample $2n\times$ data points per clock cycle, for $n \in \{4, 10, 20, 40\}$. The frequencies of the randomized clock are in the range of 7.73-12.65 Mhz.

5.5 Attack results

Table 3 presents the results of our CPA and DL-based power analysis for different oversampling factors. In the table, we show results for the MLP networks only. We also performed experiments with Convolutional Neural Networks (CNN). The results were similar to the MLPs. This is probably because we synchronized traces at the pre-processing stage.

We can see that both methods fail for the 4× oversampling case. This is due to the poor synchronization and measurement quality caused by low oversampling. For 10× oversampling, we can see that both the CPA and DL-based attacks are able to recover the subkey and, as the oversampling factor grows, fewer traces are required.

Fig. 6 shows the probability of recovering a subkey using the attack on the first round of AES after synchronizing the traces. The 4× oversampling case is omitted because the subkey cannot be recovered. Despite a slightly lower single-trace accuracy, the 20× model recovers subkeys faster than the 40× model. However, this may due to the fact that the number of tests was small.

The results of Table 3 highlight the importance of oversampling for side-channel analysis and show that ignoring oversampling may lead to an overesti-

Table 4: Comparison of CPA results with previous work.

Method	# Traces to recover full key		Key	How
	Unprotected	Protected	Enumer.	Evaluated
RFTC [JIP19]	<2K	>4M	No	not defined
DFR [HDL$^+$20]	20-30K	>1-5M	$< 2^{25}$	avg of 20 tests
Presented	<1.1K	>10M*	No	avg. of 10 tests

*For 4× oversampling factor

mation of the security of randomized clock countermeasures.

5.6 Comparison to previous DL-based attacks

Next, we compare our results to [HDL$^+$20] where a DL-based analysis based on MLP and CNN is also performed.

In one of their implementations, DFR-4-Ø, the MMCM has four output frequencies and a constant input frequency. This implementation does not have the best security/overhead trade-off, but it has a structure similar to ours. The only difference is that, in DFR-4-Ø, the switching between frequencies is done synchronously. Our implementation switches asynchronously to get a more unpredictable output clock.

According to [HDL$^+$20], DFR-4-Ø can be attacked with 1M traces using FFT-CPA, 200K traces using FFT-MLP, and 20K traces using CNN. All attacks target the last round. Considering that EM-based attacks typically need an order of magnitude more traces compared to power analysis, these numbers translate to roughly 100K for the FFT-CPA, 20K for the FFT-MLP and 2K for the CNN. Also, considering that they need approximately twice as many traces as us to attack unprotected AES, these numbers are similar to the 10× oversampling case in Table 3. Their attack use key enumeration up to $3^{16} \approx 2^{25}$ (equivalent to key guessing entropy ≤ 2) while we do not use key enumeration.

6 Conclusion

We presented a powerful side-channel attack on an FPGA implementation of AES with a randomized clock targeting the first round as the attack point. Such an approach has a greater potential to break any randomized clock countermeasure than the attacks on the last round because the effect of randomness accumulated over the multiple rounds is lost. We also demonstrated the importance of high oversampling in the security analysis of randomized clock countermeasures. Our results show that these countermeasures have a fundamental flaw. The oversampling enabled us to synchronize power traces which, in turn, made the attacks successful.

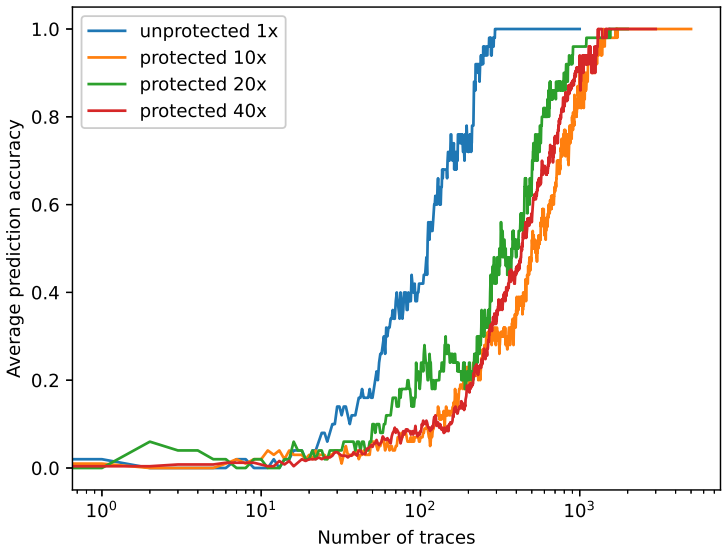

Figure 6: MLP subkey recovery success rate.

Acknowledgements

This work was supported in part by the Swedish Civil Contingencies Agency (Grant No. 2020-11632), the Swedish Research Council (Grant No. 2018-04482) and the Vinnova Competence Center for Trustworthy Edge Computing Systems and Applications at KTH Royal Institute of Technology.

References

[ACPJ17] Karim M Abdellatif, Damien Couroussé, Olivier Potin, and Philippe Jaillon. Filtering-based CPA: a successful side-channel attack against desynchronization countermeasures. In *Proceedings of the Fourth Workshop on Cryptography and Security in Computing Systems*, pages 29–32, 2017.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems*, pages 16–29. Springer, 2004.

[BHL+10] Kean Hong Boey, Philip Hodgers, Yingxi Lu, Maire O'Neill, and Roger Woods. Security of AES Sbox designs to power analysis. In

2010 17th IEEE International Conference on Electronics, Circuits and Systems, pages 1232–1235, 2010.

[BLGT05] M. Bucci, R. Luzzi, M. Guglielmo, and A. Trifiletti. A countermeasure against differential power analysis based on random delay insertion. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3547–3550 Vol. 4, 2005.

[BLOW10] Kean Hong Boey, Yingxi Lu, Maire O'Neill, and Roger Woods. Random clock against differential power analysis. In *2010 IEEE Asia Pacific Conference on Circuits and Systems*, pages 756–759, 2010.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 156–170. Springer, 2009.

[CK10] Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of CHES 2009. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 95–109. Springer, 2010.

[FW18] Dor Fledel and Avishai Wool. Sliding-window correlation attacks against encryption devices with an unstable clock. In *International Conference on Selected Areas in Cryptography*, pages 193–215. Springer, 2018.

[GM11] Tim Güneysu and Amir Moradi. Generic side-channel countermeasures for reconfigurable devices. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 33–48. Springer, 2011.

[HB10] Jip Hogenboom and Lejla Batina. Principal component analysis and side-channel attacks-master thesis. *Principal component analysis and side-channel attacks-master thesis*, pages 536–539, 2010.

[HDL+20] Benjamin Hettwer, Kallyan Das, Sebastien Leger, Stefan Gehrer, and Tim Güneysu. Lightweight Side-Channel Protection using Dynamic Clock Randomization. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pages 200–207, 2020.

[JIP19] Darshana Jayasinghe, Aleksandar Ignjatovic, and Sri Parameswaran. RFTC: Runtime frequency tuning countermeasure using FPGA dynamic reconfiguration to mitigate power analysis attacks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[JIP21] Darshana Jayasinghe, Aleksandar Ignjatovic, and Sri Parameswaran. Uclod: Small clock delays to mitigate remote power analysis attacks. *IEEE Access*, 9:108411–108425, 2021.

[KJJ] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Using unpredictable information to minimize leakage from smartcards and other cryptosystems. US Patent 6,327,661.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology — CRYPTO' 99*, pages 388–397. Springer Berlin Heidelberg, 1999.

[KJJR11] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1:5–27, 2011.

[LOM08] Yingxi Lu, Maire P. O'Neill, and John V. McCanny. FPGA Implementation and Analysis of Random Delay Insertion Countermeasure against DPA. In *2008 International Conference on Field-Programmable Technology*, pages 201–208, 2008.

[MvWB11] Ruben A Muijrers, Jasper GJ van Woudenberg, and Lejla Batina. RAM: Rapid Alignment Method. In *International Conference on Smart Card Research and Advanced Applications*, pages 266–282. Springer, 2011.

[Newa] NewAE Technology Inc. Chipwhisperer. https://newae.com/tools/chipwhisperer.

[Newb] NewAE Technology Inc. CW305 Artix Target common sources. https://github.com/newaetech/chipwhisperer/tree/develop /hardware/victims/cw305_artixtarget/fpga/common.

[Pro] ProjectVault. Verilog implementation of AES-128. https://github.com/ProjectVault/orp/tree/master/hardware/ mselSoC/src/systems/geophyte/rtl/verilog/crypto_aes/rtl/verilog.

[RBBC18] Prasanna Ravi, Shivam Bhasin, Jakub Breier, and Anupam Chattopadhyay. PPAP and iPPAP: PLL-based protection against physical attacks. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 620–625. IEEE, 2018.

[RQL19] Pieter Robyns, Peter Quax, and Wim Lamotte. Improving CEMA using correlation optimization. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 1–24, 2019.

[Sch00] Dietrich Schlichthärle. Digital filters. *Editorial Springer*, 2000.

[SDB+10] Oliver Schimmel, Paul Duplys, Eberhard Boehl, Jan Hayek, Robert Bosch, and Wolfgang Rosenstiel. Correlation power analysis in frequency domain. In *COSADE 2010 First International Workshop on Constructive SideChannel Analysis and Secure Design*, 2010.

[vWWB11] Jasper GJ van Woudenberg, Marc F Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In *Cryptographers' Track at the RSA Conference*, pages 104–119. Springer, 2011.

[Xil22] Xilinx. Vivado Design Suite 7 Series FPGA and Zynq-7000 SoC Libraries Guide (UG953). April. 2022.

[ZH08] Y. Zafar and D. Har. A Novel Countermeasure Enhancing Side Channel Immunity in FPGAs. In *2008 International Conference on Advances in Electronics and Micro-electronics*, pages 132–137, 2008.

[ZPH10] Yousaf Zafar, Jihan Park, and Dongsoo Har. Random clocking induced DPA attack immunity in FPGAs. In *2010 IEEE International Conference on Industrial Technology*, pages 1068–1070, 2010.