

Design and implementation of a waveform-matching based triggering system

Arthur Beckers, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede

KU Leuven Dept. Electrical Engineering-ESAT/COSIC and iMinds
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
`{firstname.lastname}@esat.kuleuven.be`

Abstract. Implementation attacks such as side channel attacks and fault attacks require triggering mechanisms to activate the acquisition device or fault injection equipment. Most academic works work with a very simple and reliable trigger mechanism where the device under test itself provides a dedicated signal. This however is not possible in real attack scenarios. Here the alternative is to use IO signals or coarse features of the side channel signal (co-processor switches on, power consumption goes up) for triggering. However, fault injection in particular requires very accurate timing. Our work deals with the many scenarios where such simple triggering mechanisms are not available or not effective. We present our design, architecture and FPGA implementation of a waveform-matching based triggering system. Our configurable trigger box is able to sample and match an arbitrary waveform with a latency of 128 ns. We provide results of our experimental evaluation on devices and side channel signals of different nature, and discuss the influence of several parameters.

Keywords: Triggering, waveform matching, fault injection

1 Introduction

Implementation attacks are well-known techniques that can pose a serious threat to the security of embedded devices. Side channel attacks rely on the analysis of physical observations of the device during cryptographic executions, for instance, running time [15], power consumption [16] or electromagnetic emanations [14, 17]. Fault attacks [9] on the other hand rely on injecting faults during cryptographic computations. Examples are clock glitches [7, 8], voltage spikes [19], electromagnetic pulses [13, 18] or optical attacks [5, 20].

To perform implementation attacks an adversary requires some sort of *triggering mechanism* capable of activating the side channel acquisition device or the equipment for fault injection. Depending on the concrete attack scenario, precise timing may be essential. The prevalent approach in academic works is to generate a trigger signal from within the device under test. This allows to concentrate on the evaluation of a certain attack or countermeasure, while abstracting from practical issues. This approach is however not possible in realistic

scenarios. Here, one would typically use the built-in triggering functionalities of an oscilloscope to detect simple features such as logic events in the IO line, well-defined shapes in the side channel signals (sudden amplitude changes, gaps of certain width, etc.), or a combination thereof. If the selected event does not occur just before or after the targeted operation, hold-off timers can be employed to shift the trigger closer to the time of interest.

While this approach may achieve high accuracy and reproducibility in certain settings, it suffers from two main limitations. First, the range of trigger options is limited to the capabilities of the oscilloscope, e.g. mostly edge, pulse and logic triggering. This may not be sufficient to trigger on devices where existing signals lack coarse features. And second, the insertion of hold-off timers implicitly assumes deterministic program executions. Devices with non-deterministic behaviour (due to preemptive multitasking, caches, branch predictors, etc.) or implementations with built-in countermeasures (random delays [10, 11], clock jitter, etc.) can easily make triggering a practical bottleneck.

A more suitable alternative consists in using a *pattern-based* triggering mechanism which can detect arbitrary waveforms in the side channel signals. The method runs in two stages. First, one selects a suitable *reference* or *pattern* from a window of interest in the side channel signal. And second, one employs a waveform-matching algorithm to detect such pattern on an incoming side channel signal. In the following we denote these stages as *capture mode* and *matching mode*. It is important that the analog-to-digital conversion process in both modes is the same. The selection of the underlying waveform-matching algorithm depends on the use case requirements. Fast response time is particularly desirable for fault injection attacks. Flexibility allows the method to adapt to different setups. Additionally, robustness is required to compensate for the noise that is inherently present in the signals.

To the best of our knowledge the only publicly documented solution for pattern-based triggering is icWaves [4], developed and commercialized by Riscure. This solution implements a waveform-matching algorithm based on the sum of absolute differences. It can detect pattern(s) up to 1x512 samples or 2x256 samples long and has a response time of around 500 ns. The usage of this device for laser fault injection attacks has been documented by van Woudenberg et al. in [21].

Our contribution. In this work we put forward a waveform-matching based triggering system for use in the context of implementation attacks. Our solution is specifically designed for low latency, i.e. to minimize the response time once the pattern occurs in the side channel, and it is based on an interval matching algorithm. We provide a detailed description of our design and architecture choices, as well as the implementation of a functional trigger box on an FPGA development board. Our solution supports detection of arbitrary waveforms, and incorporates multiple options to ensure flexibility and ease of adaptation to different scenarios. We illustrate these aspects by performing an empirical evaluation on two different cryptographic devices (dedicated Java Card smart card, high-speed general purpose ARM processor) with side channel signals of

different nature (power measurements using shunt resistor, contactless power measurements using EM probe). Finally, we discuss the influence of several parameters on the triggering behaviour.

2 Waveform matching

The essence of waveform matching is to compare a fixed *reference* (or *pattern*) signal g of length N samples with a continuous incoming signal h . There exist many different algorithms for pattern matching in the literature, but the vast majority work according to the same basic principle. The algorithm calculates a measure of correspondence between g and (a part of) h and represents it by a single score $T(k)$, where k represents a time shift from the starting execution point. The comparison of $T(k)$ with a pre-defined threshold determines whether the signals are considered a match. In our case, we work with discrete-time signals resulting from an analog-to-digital conversion. Therefore the threshold selection needs to account for the effect of quantization noise as well as noise caused by system and environmental variations.

In the following we review different options for waveform matching and discuss their suitability to our use case. We concentrate on algorithms that allow for low-latency matching and can be efficiently implemented in hardware. All considered algorithms perform a sample-wise comparison of the reference with the incoming signal, i.e. the score at a certain time shift k can be computed as:

$$T(k) = \sum_{m=1}^N score_m, \quad \text{where } score_m = f(g(m), h(m+k)).$$

Cross-correlation. This statistical function is perhaps the most natural algorithm to measure the similarity of two series. It uses the product of two samples to compute a measure of their resemblance as:

$$score_m = g(m)h(m+k). \quad (1)$$

Sign comparison (see Figure 1a). This method transforms the reference signal g into a binary sequence g' by assigning $g'(m) = 1$ if $g(m) > \mu$, and $g'(m) = 0$ otherwise. Here μ is the mean value of g . The incoming signal h is transformed into h' in the same manner. The sample score is calculated using g' and h' as:

$$score_m = \begin{cases} 1 & \text{if } g'(m) == h'(m+k) \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Sum of absolute differences (SAD) (see Figure 1b). This algorithm performs a sample-wise subtraction between reference and incoming signals, taking as a score the absolute value of the difference as:

$$score_m = |g(m) - h(m+k)|. \quad (3)$$

Interval matching (see Figure 1c). This algorithm defines an interval with a chosen *offset* above and below the reference. The score is calculated by checking whether the sample of the incoming signal lies within a valid interval as:

$$score_m = \begin{cases} 1 & \text{if } g(m) + offset \geq h(m+k) \geq g(m) - offset \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

Algorithm selection. Our primary selection criteria for the pattern matching algorithm is low latency, but we also consider aspects such as flexibility, implementability and suitability to our application. Despite its robustness, we discard cross-correlation due to its use of multiplications. Note that the maximum pattern length is determined by the availability of multipliers on the implementation platform, which can be quite low. Additionally, the final score $T(k)$ can grow significantly with N , posing additional demands in hardware due to need of large adders. The remaining three algorithms do not suffer from these issues, as their sample-based comparison relies on simple operations yielding low scores.

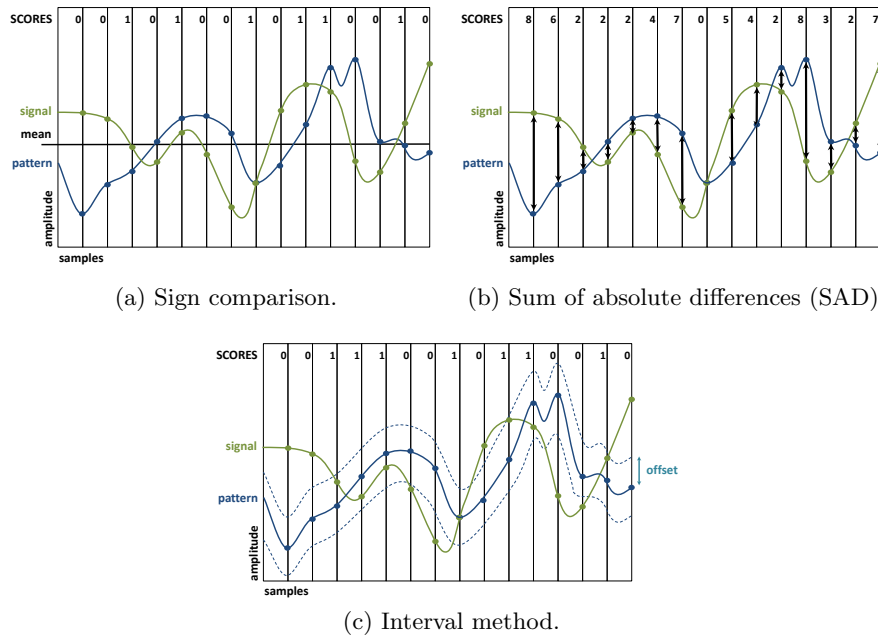


Fig. 1: Visualization of different algorithms for waveform matching.

We have ran several experiments in order to determine the suitability of all algorithms in the context of side channel signals, i.e. by testing their success rate

in matching arbitrary patterns from real (noisy) measurements collected with an oscilloscope. Our experiments showed that the sign comparison algorithm has an unreliable triggering behaviour. Therefore we opt to discard it. SAD and interval matching algorithm perform rather well, and they both have the potential for low latency and good implementability. Flexibility is thus the criteria that determines our choice. In particular, the fact that the interval matching algorithm enjoys an extra degree of configurability via the offset. An additional benefit of interval matching is a better resistance to outliers. Note that for SAD, a large difference on a single sample may have a significant impact on the $T(k)$, potentially leading to a false negative. We therefore select interval matching as core algorithm for our design.

3 Architecture

In this section we describe the hardware architecture of our triggering system based on the interval matching algorithm. The top level view is shown in Figure 2. The main components are a control unit, an analog-to-digital converter (ADC), and two modules responsible for the different modes of operation (capture and matching) sharing a memory block. The control unit provides a communication IO interface to enable external access for configuration. The incoming signal is first sampled by the ADC and then forwarded to the capture and matching modules. During capture mode, samples provided by the ADC are stored in memory when indicated by the capture signal. The amount of samples that can be stored is implementation-dependent, i.e. it is uniquely determined by the memory length. The captured measurement can be read through the IO interface. During matching mode, a pattern is first written to memory through the IO interface. The parameters of the algorithm (offset, threshold) are also externally set. The matching module contains an instantiation of the interval matching algorithm. If a match between the sampled incoming signal and the programmed reference is found, the trigger signal is activated.

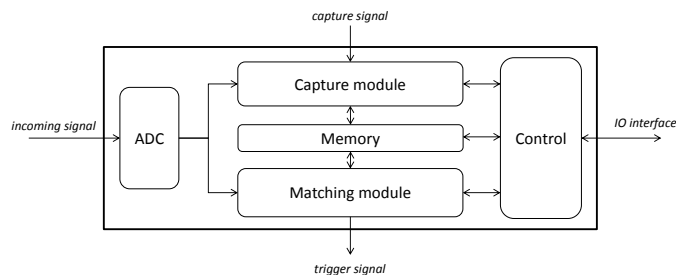


Fig. 2: Top level architecture view.

Due to its basic structure, the architecture of the capture module is not described in detail. Instead, we focus on the more critical matching module which determines the latency of the system. Its architecture, depicted in Figure 3, is essentially a shifting integration structure formed by a parallel cluster of comparator blocks and a register adder chain that shifts its content towards the threshold comparison. Each comparator block checks whether an incoming sample lies in the interval specified by the reference and the offset for $g(m)$, i.e. it performs the test described in Equation 4. The interval comparison is done in parallel for all samples in the pattern and thus we require N comparator blocks. The binary output of the comparator is fed to the register adder chain, which keeps track of the aggregated results from the previous comparisons. That way when a certain sample $h(m+k)$ is at reg_0 , the result from comparing i previous samples is already stored at reg_i . This architecture enables a quick response time when a similar pattern appears in the side channel trace. The outcome of $comparator_N$ is added to reg_{N-1} , which contains the aggregated result of the previous $N-1$ samples. The resulting $T(k)$ is directly compared with the threshold to determine whether a trigger needs to be generated, i.e. in case of a match. This full step can be implemented in one clock cycle, and it is the critical path in the module. The latency of our architecture is 4 clock cycles, since it includes buffers in the IO pins and an extra register for the flexible hold-off time before triggering.

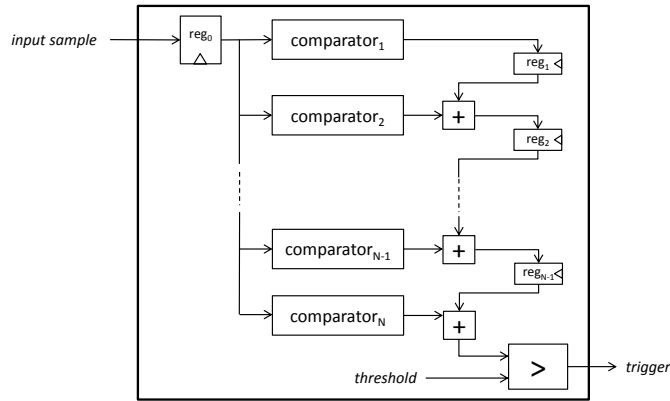


Fig. 3: Architecture of the matching module.

A more detailed view of the architecture of a comparator block and the following register adder chain is illustrated in Figure 4. An input sample stored in reg_0 is compared against the upper and lower values of the interval for a sample $g(m)$. The pair (*upper limit_m*, *lower limit_m*) is calculated in advance and stored in the corresponding registers UL_m and LL_m , i.e. UL_m contains $g(m) + offset$ and LL_m contains $g(m) - offset$. The binary output of the interval

comparisons is fed to an AND gate whose output determines whether the value in reg_{m-1} needs to be incremented before storing it in reg_m . The reset register allows to adjust the length of the reference signal, adding yet another flexibility feature to our design.

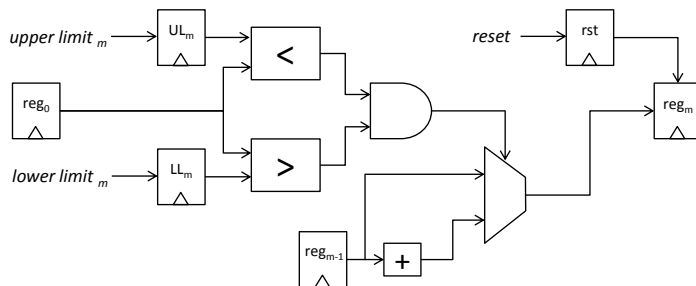


Fig. 4: Architecture of comparator block and register adder chain.

Note that the architecture given in Figure 4 is slightly different for the first and last comparator blocks. In particular, the output of the AND gate in $comparator_1$ can be directly stored in reg_1 . For the last comparator block, the output of the multiplexer goes to the threshold comparator.

4 Implementation

In this section we describe the realization of a trigger box based on the design and architecture of our waveform-matching based triggering system. We provide a brief description of the main components and interfaces and list the characteristics achieved after synthesis and place-and-route on an FPGA.

We have described our architecture in VHDL and implemented it on an Altera Cyclone IV GX FPGA Development Kit [2]. This low-end FPGA is equipped with 150k logic elements and 6.5 Mbits of embedded memory. Our hardware description deliberately avoids the use of any manufacturer-specific IP block, and thus can be easily ported to other commercial FPGAs. We have used Quartus II Web Edition Software for synthesis, place-and-route and programming. The Cyclone IV development board is not equipped with an ADC, required to sample the incoming side channel signal. Therefore we use an external Terasic AD/DA Data Conversion card [3]. This card provides two 14-bit ADC with a maximal sampling rate of 150 MS/s. It interfaces with the Cyclone IV development board via a standard HSMC interconnect header. A DC block is placed in front of the ADC to remove the DC component from the incoming signal.

We have enabled an RS-232 serial interface for IO communication and devised a rich instruction set to allow for external configuration. The commands allow at any time to select between capture or matching mode, read captured signals, program references, set the parameters of the interval matching (offset

and threshold), vary the internal sample rate, and assign hold-off times for capturing signals and/or trigger generation, among others. Both capture and trigger signals are implemented as GPIO pins. For the latter we could alternatively use one of the digital-to-analog converters (DAC) of the Terasic AD/DA card.

Synthesis results. The figures of our trigger box depend directly on the resources of the FPGA. We run the synthesis and place-and-route processes optimizing for speed. With this we obtain a design that can run at 171.17 MHz and allows for a pattern length of 1500 samples. We clock the trigger box at 125 MHz using the built-in global oscillator, therefore obtaining a latency of 32 ns for our architecture. Taking into account the 96 ns delay caused by the ADC, the total latency of our trigger box is 128 ns. Note that by using Altera’s specific IP blocks for PLLs we could generate a faster clock and thus slightly decrease our latency. The resource occupation is 133k logic elements (around 88%) and 86k flip-flops. Most resources are occupied by the 14-bit comparators. The memory depth for trace capture is only limited by the memory resources of the FPGA, which is rather large. For the purposes of testing we fix it to 60 000 samples.

Note that several tradeoffs are possible. In particular, we can increase the maximum reference length by lowering the sample resolution. Dropping the least significant bits of the ADC output from 14 to 12 bits decreases the demand of the comparators. This allows to increase the reference length to 1875 samples at the cost of some precision. Further reductions are also possible. The impact of tradeoffs on the performance of the trigger box can only be empirically evaluated, as it will naturally depend on the properties of the target device and the side channel signal.

For the sake of completeness, we list in Table 1 some features of our trigger box and the icWaves solution from Riscure. The latter figures are retrieved from the product data sheet [4]. Note that features related to signal conditioning are omitted from the listing in the table, as they will be discussed in Section 5. Other common features such as hold-off times are similarly unlisted.

	Our design			icWaves
Algorithm	interval matching			SAD
Latency	128 ns			500 ns
Sample rate	125 Msamples/s			200 Msamples/s
Resolution	14-bit	12-bit	8-bit	8-bit
Sample length	1500	1875	2625	(1 x 512) or (2 x 256)
Memory depth	60k	60k	120k	8 000k

Table 1: Main features of our trigger box designs and icWaves.

5 Evaluation

In this section we evaluate the performance of our trigger box by means of practical experiments. Our tests involve two different devices and two different types of side channel signals. We discuss the role of the interval matching parameters on the triggering behaviour and highlight the importance of analog signal conditioning.

5.1 Experimental setup

Figure 5 depicts the main components of our experimental setup as well as their interconnections. The side channel signal of the cryptographic device is connected to an oscilloscope and to the trigger box. We use the version with 14-bit resolution in all experiments. Note that the signal is modified by means of analog circuitry before being fed to the ADC of the trigger box. This is required to map the signal amplitude to the input range of the ADC, i.e. to minimize quantization errors. Additional circuitry can be used to highlight interesting features of the signal, as will be discussed in the next section. We use a computer to operate the trigger box through the serial interface.

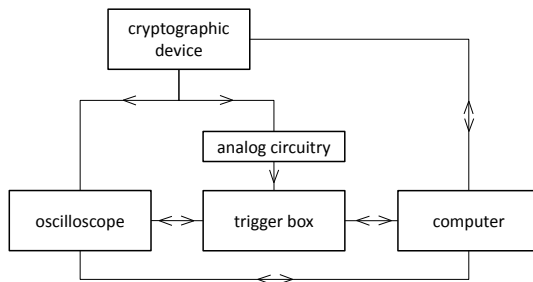


Fig. 5: Experimental setup.

Capture mode. The computer begins by configuring the trigger box to enter capture mode. It then sends a command to the cryptographic device in order to start a cryptographic computation. The oscilloscope, which monitors the side channel signal of the cryptographic device, is responsible to indicate to the trigger box when to start recording. The incoming signal is then stored in internal memory (60 000 samples) and sent to the PC.

Note that in our setup the oscilloscope is responsible for activating the trigger box during capture mode. Hence it is required to (at least once) be able to configure the oscilloscope to trigger near the window of interest. This step can be done with the usual techniques, e.g. by detecting simple features on the IO line or on the side channel signal. The activation signal is provided by the AUX out port of the oscilloscope. An optional hold-off time can be specified either in

the oscilloscope or in the trigger box. We stress that this step only needs to be executed once. During our experiments, we have always been able to carry it out without major difficulties, even if the trigger programmed in the scope has an unstable behaviour.

Matching mode. The first step in matching mode consists in selecting a pattern from the pre-recorded trace. Any software visualization tool can be used for this, e.g. Matlab. The pattern is sent to the trigger box along with the parameters of the interval matching algorithm (offset and threshold). Once the trigger box is configured, the computer sends a command to the cryptographic device to start the cryptographic operations. If a match is found the trigger box generates a pulse on the trigger signal, optionally with a certain hold-off time. In our experimental setup this signal is connected to an analog channel of the oscilloscope. This allows us to easily verify whether the trigger box has found a matching pattern and, if so, whether it is the correct one. We note that in a real attack scenario the output trigger signal will be directly connected to an acquisition device to start measurement collection, or alternatively to some equipment for fault injection.

5.2 Experiments with a Java Card smart card

Our first evaluation of the trigger box is performed on a Java card that contains an applet to compute RSA signatures. The details of the implementation are unknown to us. We access the side channel power consumption of the Java card by measuring the voltage drop over a shunt resistor placed in the ground line. An exemplary trace collected with an oscilloscope at 80 MS/s is shown in Figure 6 (top). The two long similar patterns indicate that the implementation uses RSA-CRT. Zooming into either of the CRT-branches reveals repetitive patterns corresponding to the inner modular operations, i.e. square and multiply. This is illustrated in Figure 6 (bottom).

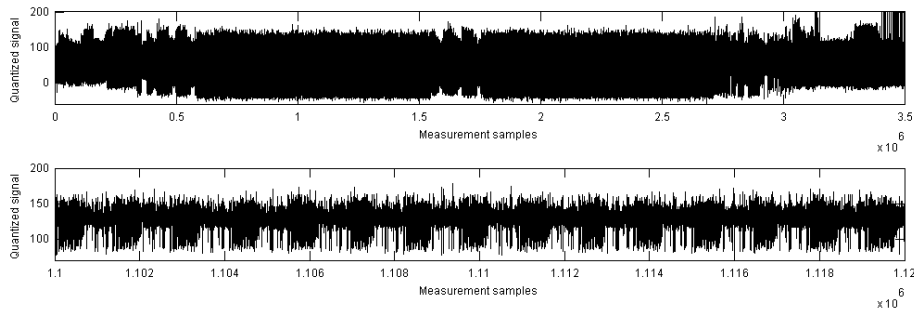


Fig. 6: Power measurements of the RSA signature applet in the Java card. Full execution (top), zoom into first CRT-branch (bottom).

Single matching. We begin our experiments by bringing our setup to capture mode and configure it to record a trace close to the beginning of the first CRT-branch. The aim is to find a unique pattern that appears only *once* per cryptographic execution. We set the sampling rate of the trigger box at 125 MS/s. An exemplary recorded trace of 60 000 samples is shown in Figure 7 (left). Note that the shape of the trace in Figure 7 differs from the one in Figure 6. This is due to the ADC in the trigger box being different from the ADC in the oscilloscope.

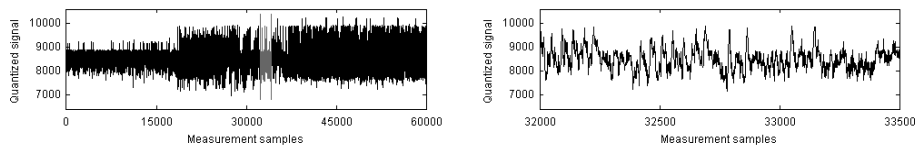


Fig. 7: Trace recorded by the trigger box during capture mode, pattern selected for matching mode in grey (left), zoom into the pattern (right).

The next steps are to select a pattern and to set the parameters of the interval matching algorithm. The behaviour of the trigger box will naturally depend on a good combination of these variables. Our goal is to obtain a single correct trigger per execution while avoiding (or at least minimizing) false negative and particularly false positives.

It is important to stress that there is no generic rule on how parameters need to be set. However, some general observations can be made. First and most important, it is critical that the pattern we want to trigger on is as unique as possible. Since the length of the reference is limited, a way to incorporate more unique features is by varying the sample rate. This variation should however not result in an undersampling of the signal. Second, using the maximal pattern length is in general beneficial for detection. However, once the unique features are incorporated in the pattern it may not make a difference in terms of success rate. We have empirically observed the existence of a convergence point in which the percentage of correct matches stops varying even when the reference length is increased. Third, there is a natural relation between signal noise and offset. Noisy side channel signals will inevitably demand higher offsets to account for the variability in the observations. This comes however at a risk, as arbitrarily high offsets will cause wrong patterns to be considered a match. Finally, the threshold can be used as a tuning parameter once all other values are constant. In fact for a given pattern and offset there will be a threshold value that maximizes the percentage of correct matches.

Let us illustrate this behaviour with an example. Assume we select as pattern the grey area in Figure 7 (left) which is also depicted in Figure 7 (right). We set the pattern length to the maximum (1 500 samples) and test different offset values. These are rather conservative, as the noise level in our side channel signal is quite low. Figure 8 shows an area plot that visualizes the responses of the

trigger box for each offset value in function of the threshold. The vertical axes shows the outcome (in percentage) computed by running multiple experiments. Each color (grey, black, white) indicates a possible outcome of the experiments (false positive, correct match, false negative).

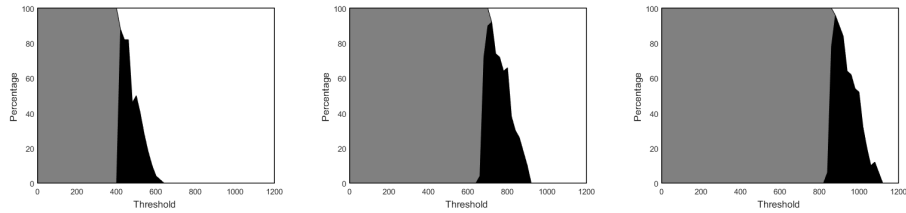


Fig. 8: Percentage of outcomes for varying thresholds: grey (false positive), black (correct match), white (false negative). Values of offset are 150 (left), 300 (middle), 450 (right).

The plots clearly show that for a given offset the desired triggering behaviour can be tuned by varying the threshold. In most cases one wants to avoid false positives. Therefore, it is preferable to select a threshold that is slightly higher than the one that maximizes the percentage of correct matches, i.e. to move away from the grey area in the plot. We also observe that higher offsets demand higher thresholds in order to avoid false positives. In contrast, the maximal percentage of correct matches only varies slightly with the offset.

Our best results when triggering at the beginning of the first CRT-branch are obtained when selecting the combination offset of 450 and threshold of 900, which yields a percentage of correct matches of 90%, with 10% false negatives and no false positives.

Multiple matching. Our follow-up experiment consists in checking whether it is possible to trigger on *each* execution of a modular operation in the CRT-branches. For this we need to select the repetitive pattern in Figure 6 (bottom), which perfectly characterizes the occurrence of squarings and multiplications. We lower the sampling rate of the trigger box to 62.5 MS/s in order to fit the pattern in less than the maximum 1 500 samples supported by our implementation. The recorded pattern is shown in Figure 9 (left).

Although the pattern appears to contain characteristic features, it turns out to be completely unsuitable for the purposes of triggering. In fact, all our experiments yield a negligible percentage of correct matches for any combination of offset and threshold. Most observed outcomes are false negatives or false positives. The reason behind these results is the repetitiveness of the features in the reference, which is caused by the relatively high frequency of the side channel signal. This causes a non unique pattern with many matches in the incoming signal.

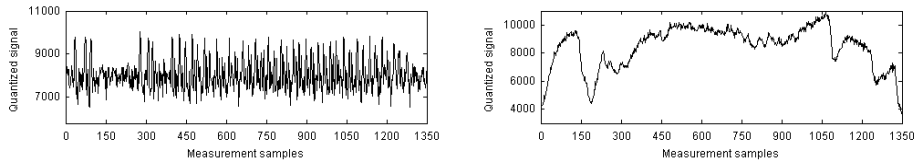


Fig. 9: Repetitive pattern in CRT-branches. Normal (left), with envelope detection (right).

In order to overcome this issue, some extra analog circuitry is required to condition the signal before the ADC and highlight its low-frequency features. We opt to incorporate an ADL5511 envelope detector board with some added capacitors that effectively reduces the bandwidth of the incoming signal to 2 MHz. The envelope detector consists of a rectification stage followed by a low-pass filtering. Using this circuitry we can effectively reduce the bandwidth of the signal independently of its high frequencies, and therefore capture useful features.

We have repeated our experiments by using an envelope detector in combination with an extra 30 dB amplifier (Langer PA 303) to minimize quantization noise. The newly selected pattern, shown in Figure 9 (right), corresponds to the envelope detected signal of a modular operation. By using this configuration with an offset of 50 and a threshold of 810, we are able to increase our success rate to nearly 100% without any false positive, i.e. our box is able to trigger on almost every occurrence of a modular operation in a single RSA-CRT operation. In fact, the only false negatives correspond to the first operation in each CRT-branch, which have a shape different than the rest.

5.3 Experiments with an ARM processor

For our second evaluation of the trigger box we switch to a more challenging platform featuring non-deterministic program execution. We use a BeagleBone Black [1] equipped with a Sitara ARM Cortex-A8 32-bit RISC processor. The non-deterministic behaviour of the processor stems from its dynamic branch predictors, L1 and L2 cache memories and out-of-order execution engine. Additionally, the processor runs a preemptive multitasking Linux operating system with e.g. scheduling, context switches and interrupts. The core runs at a clock frequency of 1 GHz.

In order to monitor the power consumption, we place an electromagnetic pen probe on one of the decoupling capacitors as previously described in [6]. Due to the low output voltage range of the pen probe and the high operating frequency of the processor, we require some extra analog signal conditioning. We insert a 30 dB amplifier (Langer PA 303) between the pen probe and the input of the envelope detector, followed by another 30 dB amplifier used to minimize the quantization noise in the ADC.

Our test application is a software implementation of the Advanced Encryption Standard (AES) [12] that performs bulk encryption. Finding a consistent

and reliable trigger on the side channel signal is an arduous task when using only an oscilloscope, i.e. sudden amplitude changes occur frequently due to other running processes causing multiple false positives. However, we only need one correct trigger in order to store a measurement during capture mode.

A trace recorded by our trigger box is shown in Figure 10 (left). Thanks to the envelope detector, the batched executions of the AES are identifiable in the trace (30 encryptions in this particular case). We select a pattern corresponding to a single execution, depicted in Figure 10 (right), and program it into the trigger box. The length of this reference signal is 960 samples, and it is obtained with a sampling rate of 25 MS/s. The lower sampling rate is an additional benefit from using an envelope detector, as it allows to sample the now low-frequency signal without losing its features.

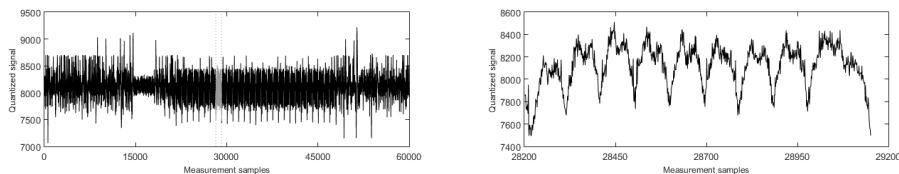


Fig. 10: Trace recorded by the trigger box during capture mode, pattern selected for matching mode in grey (left), zoom into the pattern (right).

By setting the offset to 130 and choosing a threshold of 630, we achieve 99.2% of correct matches, i.e. we trigger on almost all AES executions, without any false positive. The remaining 0.8% of false negatives correspond to the first encryption and any other execution occurring after an interrupt. This is an artifact of the instruction cache of the processor, which is filled with code after every context switch. The cache filling leads to a different shape in the side channel signal, which the trigger box correctly identifies as no match.

It is worth mentioning that our success rate is kept even when we increase the number of batched encryptions to e.g. 50 000. The same outcome is achieved when we increase the CPU load to 100% by enabling heavy computational tasks in parallel such as RSA key generation in OpenSSL.

6 Conclusions

Triggering is critical to enable implementation attacks in real scenarios. The lack of accurate and reliable trigger points may fully prevent to mount an attack on a certain implementation, even if unprotected. This is particularly true for modern high-end devices with non-deterministic behaviour due to complex CPU architecture and operating system. In this work we have documented the design, architecture, implementation and evaluation of a waveform-matching based triggering system tailored to the context of embedded security. With this, we put forward a tool that can be of use to the research community.

Acknowledgements. We would like to thank Victor Förster for initial contributions to the system’s design and architecture. This work was supported in part by the Research Council KU Leuven: C16/15/058. In addition, this work was supported by the Flemish Government, FWO G.0550.12N, by the Hercules Foundation AKUL/11/19, and through the Horizon 2020 research and innovation programme under grant agreement 644052 HECTOR. Benedikt Gierlichs is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO).

References

1. BeagleBone Black Starting Guide. Beagleboard.org, Last accessed: December 2015. <http://beagleboard.org/getting-started>.
2. Cyclone IV GX FPGA Development Kit. Altera, Last accessed: December 2015. https://www.altera.com/products/boards_and_kits/dev-kits/altera/kit-cyclone-iv-gx.html.
3. Highspeed AD/DA Card. Terasic, Last accessed: December 2015. <http://www.terasic.com.tw/cgi-bin/page/archive.pl?No=278>.
4. icWaves Datasheet. Riscure, Last accessed: December 2015. <https://www.riscure.com/security-tools/hardware/icwaves>.
5. M. Agoyan, J. Dutertre, A. Mirbaha, D. Naccache, A. Ribotta, and A. Tria. How to flip a bit? In *IOLTS 2010*, pages 235–239. IEEE Computer Society, 2010.
6. J. Balasch, B. Gierlichs, O. Reparaz, and I. Verbauwhede. DPA, Bitslicing and Masking at 1 GHz. In T. Güneysu and H. Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 599–619. Springer, 2015.
7. J. Balasch, B. Gierlichs, and I. Verbauwhede. An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, editors, *FDTC 2011*, pages 105–114. IEEE Computer Society, 2011.
8. H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer’s apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, Feb 2006.
9. D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In W. Fumy, editor, *EUROCRYPT ’97*, volume 1233 of *LNCS*, pages 37–51. Springer, 1997.
10. C. Clavier, J. 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, 2000.
11. J. 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, 2009.
12. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.
13. A. Dehbaoui, J. Dutertre, B. Robisson, and A. Tria. Electromagnetic transient faults injection on a hardware and a software implementations of AES. In G. Bertoni and B. Gierlichs, editors, *FDTC 2012*, pages 7–15. IEEE Computer Society, 2012.
14. K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer, 2001.

15. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Kobitz, editor, *CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
16. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
17. J. Quisquater and D. Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In I. Attali and T. P. Jensen, editors, *E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.
18. J.-J. Quisquater and D. Samyde. Eddy current for Magnetic Analysis with Active Sensor. In *Esmart 2002*, pages 185–194, 2002.
19. J. Schmidt and C. Herbst. A practical fault attack on square and multiply. In L. Breveglieri, S. Gueron, I. Koren, D. Naccache, and J. Seifert, editors, *FDTC 2008*, pages 53–58. IEEE Computer Society, 2008.
20. S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In B. S. K. Jr., Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 2–12. Springer, 2002.
21. J. G. J. van Woudenberg, M. F. Witteman, and F. Menarini. Practical optical fault injection on secure microcontrollers. In L. Breveglieri, S. Guilley, I. Koren, D. Naccache, and J. Takahashi, editors, *FDTC 2011*, pages 91–99. IEEE Computer Society, 2011.