Making and Breaking Leakage Simulators

Jake Longo Galea, Daniel Martin, Elisabeth Oswald, Daniel Page, Martijn Stam

Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, United Kingdom.
{jake.longo, dan.martin, elisabeth.oswald, dan.page, martijn.stam}@bris.ac.uk

Abstract. Recently, Standaert *et al.* (Crypto'13) advocated the notion of simulatable leakage as a means to connect theoretical leakage resilience to practice. They argued that using simulators based on actual physical devices, the assumptions underlying their proofs of side channel resistance become empirically 'verifiable' as evaluation labs can scrutinise the indistinguishability of the simulator by actually 'playing' the games that involve real versus simulated leakage. Standaert *et al.* proposed a concrete, block cipher based instantiation of a leakage resilient pseudorandom generator. They provided a high level definition of a simulator based on splicing two partial traces, and included detailed reasoning why their simulator (for AES-128) would resist state-of-the-art side channel attacks.

We exhibit a distinguisher against their simulator, thereby falsifying their hypothesis. We demonstrate the efficacy of our distinguishing technique by experimental validation using concrete implementations of the Standaert *et al.* simulator on several different platforms. Our successful analysis is based on 'tracking' consistency (and likewise spotting simulator inconsistencies) in leakage traces by means of cross correlation. By taking the cross correlation between trace points, we can estimate real-or-simulated based either on a single key that is used multiple times, or based on multiple runs of Standaert's *et al.* security game with varying keys each used only once. Since the game hybridizes (in the number of keys used), the latter implies that theoretically our distinguisher already wins when a single key is used with a single trace of side channel leakage!

Finally, we propose several alternative simulators, based on splitting traces at points of low intrinsic cross-correlation, which are more promising w.r.t. the cross-correlation distinguisher. Unfortunately, these new simulators come with significant caveats, and we conclude that the most natural way of producing simulated leakage is by using the underlying construction 'as is' (but with a random key). Provided the actual implementation has a low signal-to-noise ratio, we believe it practically infeasible to distinguish between real and simulated traces: when only a few very noisy leakages are made available to an attacker, signal processing techniques that rely on having sufficient observations are not applicable.

Keywords: cross correlation, simulatable leakage, leakage resilience, side channel attacks

1 Introduction

Practical side channel security and formal leakage resilience are at odds: practitioners concentrate on ensuring security properties for particular implementations on specific devices. Theorists value theoretically sound, sometimes elegant, but in any case mathematical arguments that apply to schemes when considering certain classes of leakage functions. Unfortunately, the many offerings that are now available from leakage resilient cryptography do not make their way into the realm of practical side channel protections. Theoretical leakage resilience is viewed as too high level, expensive and disconnected from practice, e.g. (to the best of our knowledge) no practical method exists that would actually describe how to derive a leakage rate (to connect it to the works of Dziembowsky and Pietrzak [5]), or a leakage bound (to connect it to the bounded retrieval model [1]) for a concrete device with real implementations of cryptography. In addition the insufficient restrictions on what a leakage function can do (they are typically modelled as polynomial time functions with only some bound on their output range [14]) allows the adversary to play 'magic' tricks with them, such as the infamous future computation attack [5]. Whilst from a theoretical perspective this might be considered elegant, the consequences of having an unrealistically powerful adversary leads to constructions that are effectively discarded by the practical community.

1.1 What is leakage?

The fundamental discrepancy between theory and practice lies in the different understanding of what constitutes ‘leakage’. When examining the vast literature on side channels and leakage resilience, there seem to be two different ideas behind the concept of a leakage function. The first idea can typically be found in works written by practitioners (such as [9,11]) and centre around a mathematical description of the physical nature of real-world leakage traces. The second idea, in contrast, seeks to define leakage functions in more general and powerful terms, and is often found in theoretical contributions.

Leakage understood as the modelling of the physical nature of the observed leakage points. The first understanding of ‘leakage function’ is that of it being the mathematical function that describes the shape and form of points in leakage traces. Such a function then models the manner in which the operations and data act on the physical environment, alongside other electrical components including the measurement apparatus, and the environment conditions. This understanding of leakage implies that the leakage function fundamentally depends on how the leakage is acquired (because it includes the measurement apparatus). It also implies that the leakage function, whilst being key dependent, is in principle unbounded: every new measurement gives more information. In some works, notably [4], the concept of ‘exploitable’ leakage is introduced. Here the conceptual emphasis is that the term ‘leakage’ refers to leakage about the key. Because a key has finite length, the leakage function can never reveal more than that amount of information. It is hence a function that could (depending on the number of queries to the function), under ideal circumstances, reveal the entire key information but no more than that.

Leakage understood as a mathematical concept largely separated from any physical interpretation. The second understanding is that leakage is a function that has certain restrictions (see [5,6]) such that defining cryptography is still possible, but otherwise is meant to be as general and powerful as possible. Consequently a direct physical interpretation is not intended as such; rather, the idea is pursued that any ‘realistic’ leakage function is included given the general nature of the definition. The discrepancy that arises from this perspective is that for practitioners, any overhead that is incurred because of ‘proof relics’ or protection against ‘magic’ such as future computation attacks is a unnecessary (and often unaffordable) expense.

1.2 Simulatable leakage

The recent contribution by Standaert *et al.* [19] hence comes as a welcome addition to the current approaches of dealing with the concept of leakage as part of provable security. In a nutshell, the authors suggest that a sensible notion for leakage resilience is that if real leakage cannot be distinguished from simulated leakage (from a simulator that does not have access to the secret key k), it cannot contain any information about the key. This approach removes the problem of having to mathematically define a leakage function: instead one gets a concrete instance of it in the form of an *actual* simulator. Hence rather than struggling with meaningful definitions for what is leakage, and how to practically derive bounds for it for a concrete device, the new challenge is to define and build (practical) simulators.

The challenge of building concrete leakage simulators for invertible functions was also taken up in [19], where the authors suggest an efficient solution: given some public input x and output y of an invertible scheme f, they explain how a trace can be constructed with a random key k^* that is consistent with the public inputs x and y. This can be done by choosing a random key k^* and computing $y\prime = f_{k^*}(x)$, $x\prime = f_{k^*}^{-1}(y)$, and then $y = f_{k^*}(x\prime)$ to generate leakage traces $L(x) = l^l||\alpha$, $L(x\prime) = \beta||l^r$ that can be ‘split up’ (as indicated) and concatenated to a new simulated trace $(l^l||l^r)$. The $q-\text{sim}$ game, that can be played in practice, consists of the attacker trying to distinguish real traces from simulated traces given

q real (or simulated) traces by using whatever state of the art attacks that are available. The rest of [19] consists in discussing why state of the art side channel attacks cannot win this game effectively as well as using the game restricted to $q = 2$ to prove a PRG construction leakage resilient in the standard model.

1.3 Our contribution

We show that there exists a side channel distinguisher that can effectively distinguish simulated traces by detecting that they are split and concatenated in the inner encryption rounds. We do not require knowledge of input or output, or access to the auxiliary leakage oracles for building templates. Our attack is based on using cross-correlation to check the consistency of the data flow across encryption rounds and in order to pinpoint inconsistencies from splitting and concatenating traces, and works across different real world platforms. Our distinguisher has the property that playing a game with q traces for a single key is equivalent to playing a game with one trace for q keys. This implies that for our distinguisher the game hybridises over the keys (but not over q, see our generalisation the $p-q-$sim game for details).

We analyse the properties of (cross-)correlation in this specific application to identify which factors impact on the ability to win the game efficiently. The factors are the (intrinsic) cross-correlation between points in leakage traces, and the ratio between signal and noise. Whilst changing the ratio between signal and noise is well understood and practically achievable, it is not sufficient with regards to decreasing an adversary's chance to win the $p-q-$sim game. Our attempts to work with points that have low intrinsic cross-correlation were only successful in theory: for the concrete instantiation in our paper, which is based on AES, there exist such points because of the nature of SubBytes. Theoretically, the input and output of the SubBytes operation would be good candidates and we explain why in practice it remains a challenge to exploit this. Finally we suggest a method that indeed withstands the powerful cross-correlation distinguisher, which is based on a double block cipher construction. Our proposed simulator then uses a meet-in-the-middle attack to determine keys that map x to y without introducing inconsistencies in the data flow. The ratio between signal and noise then determines how 'easy' it is to attack this construction with conventional side channel methods.

Given that even the computationally expensive simulator still requires noisy traces, it would seem that the most natural way to produce simulated leakage is to just use the construction 'as is', but run it with a random key. For sufficiently noisy traces even profiling prior to the game will not help an attacker: noisy leakage implies that an adversary must have sufficiently many traces to distinguish real from simulated leakage. By limiting q in the game stage this will be infeasible. With this somewhat simple fact in mind, we can argue that leakage can be simulated for any cryptographic primitive or construction. It however requires implementations with high noise.

Outline. We review the 'split and concatenate' simulator in Sect. 2 and explain and analyse our attack in Sect. 3. Thereafter in Sect. 4 we discuss approaches for 'clever' fixes to the split and concatenate simulator. We provide information about the practical aspects of our setup and devices in App. B.

2 Simulatable leakage: Standaert *et al.*'s model

Before we discuss the model introduced by Standaert *et al.* in [19], we will introduce the required notation. The probabilistic leakage of a block cipher will be given as $BC_k(x) \rightsquigarrow l \overset{\text{def}}{=} L(k, x)$ where L is the leakage function, x is the plaintext and k the secret key. The leakage function can be described as a vector $l = (l_1, l_2, ...)$. For a block cipher, which typically consists of several encryption rounds, we group those trace points corresponding to a round and indicate this by placing the round number as a superscript l^i. For AES (short for Advanced Encryption Standard) we can represent a leakage as $l = [l^1, \ldots, l^{10}]$.

```
**Experiment** $q-\mathrm{sim}(A, \mathbf{BC}, L, S^L, b)$:
$k \xleftarrow{\$} \{0,1\}^n$
$k^* \xleftarrow{\$} \{0,1\}^n$
$i \leftarrow 0$
$j \leftarrow 0$
$l \leftarrow 0$
$b' \leftarrow A^{Enc,(\cdot),Gen(\cdot,\cdot),Leak(\cdot,\cdot)}()$
Return $b\prime$
```

```
**proc** $Enc(x)$:
$i \leftarrow i + 1$
**if** $i > q$ **then**
    Return $\perp$
**end if**
$c \leftarrow \mathbf{BC}_k(x)$
**if** $b = 0$ **then**
    $\Lambda \leftarrow L(k, x)$
**else**
    $\Lambda \leftarrow S^L(k^*, x, c)$
**end if**
Return $(c, \Lambda)$
```

```
**proc** $Gen(z, x)$:
**if** $l = 1$ **then**
    Return $\perp$
**end if**
$l \leftarrow 1$
**if** $b = 0$ **then**
    $\Lambda \leftarrow S^L(z, x, k)$
**else**
    $\Lambda \leftarrow S^L(z, x, k^*)$
**end if**
Return $\Lambda$
```

```
**proc** $Leak(z, x)$:
$j \leftarrow j + 1$
**if** $j > s_{\mathcal{A}}$ **then**
    Return $\perp$
**end if**
$\Lambda \leftarrow L(z, x)$
Return $\Lambda$
```

Fig. 1. q-simulatable leakage from [19].

We will later require to split and concatenate leakage vectors. For this purpose we use the short-hand $l^{i,j} = [l^i, \ldots, l^j]$ to denote that we take the parts of the leakage vector that correspond to rounds i up to j. To highlight where we 'split and concatenate' within leakage vectors we use $||$ to explicitly mark out concatenations. We often need to work with sets of leakages and so denote such a set with bold typesetting, i.e. $\mathbf{l}$ now is a set of leakages, $\mathbf{l}^{i,j} = [\mathbf{l}^i, \ldots, \mathbf{l}^j]$ now means that we refer to rounds i until j in all leakages in the set. Finally if we need to differentiate between points within multiple leakages we will use a subscript, i.e. $\mathbf{l}_u = (l_{1,u}, l_{2,u}, \ldots)$ means we index the $u-$th point in each leakage vector in $\mathbf{l}$. Now that the notation has been defined, we are ready to discuss the model.

2.1 Model and $q - \mathrm{sim}$ Game

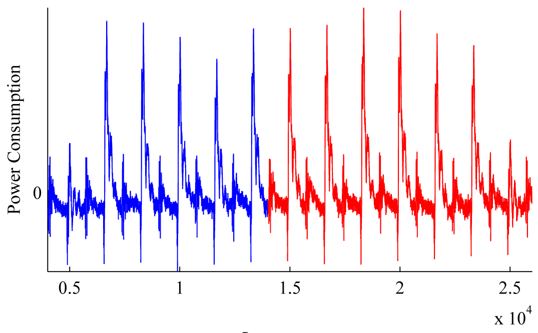
In Fig. 1 we describe the q-simulatable leakage game from [19] for completeness. Recall that the intuition captured in the $q - \mathrm{sim}$ game is that if an adversary cannot distinguish real (i.e. depending on the secret key) from simulated (i.e. depending on a random key) traces, the real traces cannot contain any information about the secret key. In the game, the adversary can make q queries to the Enc oracle and receives back the encryption of x under key k and either the real leakage $L(k, x)$ or the simulated leakage $S^L(k^*, x, c)$. The adversary can also make $s_{\mathcal{A}}$ queries to the leakage oracle with a chosen key and message, which represents profiling a device (i.e. the adversary can attempt to derive (compute, or represent) the otherwise unknown leakage function). We note that this, in particular, allows an adversary to query the leakage function on the inputs from the game, so templates specifically for the inputs used in the game can be derived. The last oracle which can be called once is Gen which delivers simulated leakage for a chosen message/key pair where either the real or random key in the game is output as the ciphertext. This is to represent the fact that often encryption keys themselves are the result of block cipher invocation, i.e. in practice (and in the constructions discussed in [19]) the encryption key used in round r is generated as the output of the block cipher in the previous round $r - 1$. The adversary's advantage is calculated as $\mathbf{Adv}^{q-\mathrm{sim}}_{L,S^L,\mathbf{BC}}(A) = |\Pr[q - \mathrm{sim}(A, \mathbf{BC}, L, S^L, 1) = 1] - \Pr[q - \mathrm{sim}(A, \mathbf{BC}, L, S^L, 0) = 1]|$.

2.2 Simulator

For completeness we also recall the simulator in Fig. 2(a). The simulator takes in a random key k^* as well as a plaintext/ciphertext pair (x, c); note that this pair was created using a key different to k^*. The simulator first encrypts m under k^* and records the leakage. The next step is to decrypt c under k^* to get a new plaintext x'. The final stage is to encrypt x' under k^* (note that this will encrypt to c) and record the leakage. The leakage is the split (in half) and concatenated such that the first part of the new trace

```
simulator $S^L_{s\&c}(k^*, x, c)$:
$c' \leftarrow \mathbf{AES}_{k^*}(x) \rightsquigarrow l^{1,5}||\alpha$
$m' \leftarrow \mathbf{AES}^{-1}_{k^*}(c)$
$c \leftarrow \mathbf{AES}_{k^*}(x') \rightsquigarrow \beta||l^{6,10}$
Return $l^{1,5}||l^{6,10}$
```

(a) Description of simulator $S^L_{s\&c}$.



(b) Simulated leakage $S^L_{s\&c}$ based on SASEBO-R power traces.

Fig. 2. Definition for the $S^L_{s\&c}$ simulator and an exemplary trace.

corresponds to the leakage on x while the second half corresponds to the leakage on c. This simulator is referred to as split and concatenate ($S_{s\&c}$) simulator and we will refer to traces from this simulator as $s\&c$-traces.

3 Breaking the split and concatenate simulator

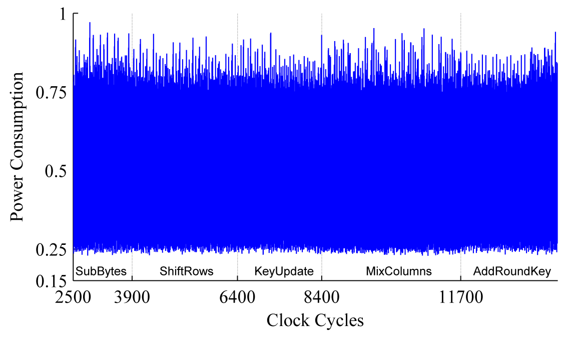
Whilst the proposed simulator can be instantiated with any invertible function, we continue to follow Standaert *et al.*'s exposition and use implementations of AES as running examples in our paper. To explain why cross-correlation is an efficient distinguisher we briefly recall how a typical side channel trace relates to the information flow in an AES implementation.

3.1 Properties of real world leakage

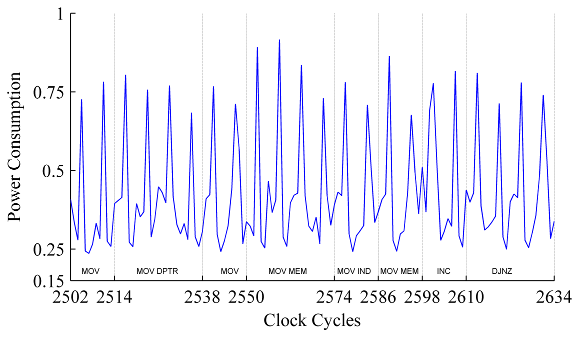
As argued in previous work [9,4], any implementation of AES will happen as a sequence of steps that lead to the processing of intermediate values. For instance, in a serial implementation, the state bytes will be touched sequentially and updated according to the AES round function. As all high level functions (SubBytes, MixColumns, etc.) are processed by some gates at the lowest level, a clock signal is involved that governs (to some extent) when data flows between gates. All changes in gates, in each clock cycle, produce some form of leakage (time for signals to travel, power consumed, radiation emitted) that becomes available through observing the device. Fig. 3 illustrates the power consumption measurements from an AES implementation on a simple 8-bit microcontroller (i.e. all intermediate values are represented as bytes). Evidently, we can see patterns in Fig. 3(a), which represents a single AES round.

In case of this very simple processor, we can even identify the effect (w.r.t. shape and height) of individual instructions in the power consumption by zooming into the trace further, see Fig. 3(b). It shows now only a small part of the first round that corresponds to performing the SubBytes operation. The instructions used for this purpose are register transfers, (MOV and MOVC), a register increment (INC), and conditional branches which correspond to a loop that runs through the 16 state bytes.

In a parallel implementation, each opertaion will touch multiple state bytes but the sequence of round functions must remain sequential, see Fig. 10 (at the end of App. B) for the high-level description of the AES implementation on the SASEBO-R. SubBytes is sometimes implemented using combinational logic in dedicated hardware [22]. Although combinational logic is not clocked, typically at the end of a combinational path some storage elements are placed which becomes synchronized with the clock signal. Referring to Fig. 4(c) we can see that there are 10 visible rounds.



(a) Power trace showing a single AES round.



(b) Power trace showing individual assembly instructions.

Fig. 3. Power traces for an 8051 8-bit microcontroller.

3.2 Cross-correlation as a distinguisher

The term correlation is often used to refer to a broad class of statistical dependencies in data. There are different metrics to measure correlation. Most commonly used (at least in side channel attacks) is the Pearson correlation coefficient, and we use this metric when we refer to correlation in this article. In the context of side channel analysis, correlation has been applied in DPA before [3] and its properties as a good distinguisher are well understood [12].

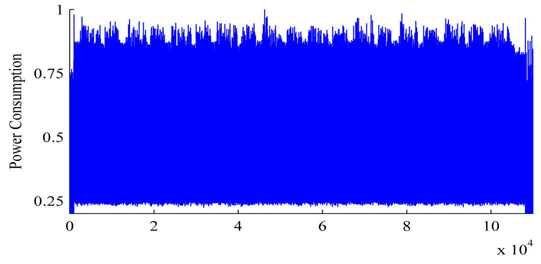
Cross-correlation is a term coined in signal processing and is commonly used to identify (and measure) similarities of wave forms. It has been used in the context of side channel analysis before, e.g. [13,18,21].

We make the simple observation that the cross-correlation of signals of length one (i.e. points) equates to computing the correlation between trace points $(\mathbf{l}_u, \mathbf{l}_v)$ (as opposed to the correlation with key-dependent predictions in the DPA context). Equation (1) recalls how (Pearson) correlation is defined and estimated.

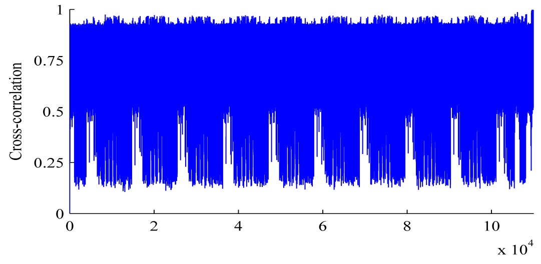
$$\begin{aligned} xr(\mathbf{l}_u, \mathbf{l}_v) &= \frac{Cov(\mathbf{l}_u, \mathbf{l}_v)}{\sqrt{Var(\mathbf{l}_u) \cdot Var(\mathbf{l}_v)}} \\ &= \frac{\sum_i (l_{i,u} - \bar{\mathbf{l}_u}) \cdot (l_{i,v} - \bar{\mathbf{l}_v})}{\sqrt{\sum_i (l_{i,u} - \bar{\mathbf{l}_u})^2 \cdot \sum_i (l_{i,v} - \bar{\mathbf{l}_v})^2}} \end{aligned} \tag{1}$$

Note that we estimate the cross-correlation between trace points, which are a function of x and k. Consequently, the estimator xr is computed with respect to the distributional variation of those inputs. It is irrelevant whether we vary x and k jointly or independently, it simply holds that more leakage traces lead to a better estimation. In particular, this means that we can estimate xr by varying k, which has an important implication for the $p - q -$ sim game.

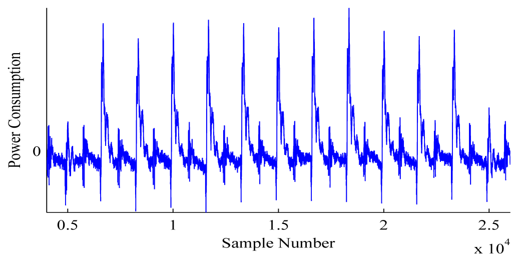
Cross-correlation traces. We explained before that cryptographic algorithms are implemented as step-wise processes (with varying degrees of parallelism). Although AES mixes key and input efficiently, and so the correlation between key, input, and output is small, we can expect a high correlation between the subsequent states, e.g. we expect a high correlation between the input and output of AddRoundKey, as well as states that operate on the same data (even though they might be separated in time). This in turn implies that we can expect a high correlation between subsequent points within leakage traces, as well as points that are related to the same intermediate values. Further to that we can also expect high



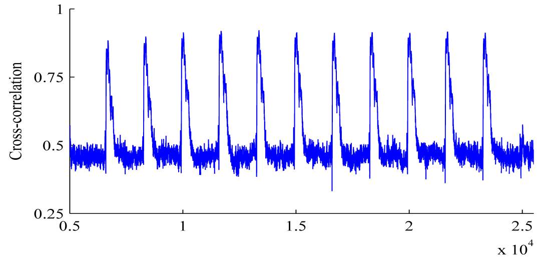
(a) AES power trace (8051).



(b) Cross-correlation trace (8051).



(c) AES power trace (SASEBO-R).



(d) Cross-correlation trace (SASEBO-R).

Fig. 4. AES power traces and cross-correlation plots.

correlations between data that is related to the program state but independent of the states, e.g. the value of the program counter, pointers to memory locations, etc. Hence any implementation will lead to a specific cross-correlation trace depending on the data flow.

By producing a cross-correlation trace that shows the cross-correlation of *all* pairs of points, i.e. $\{xr(\mathbf{l}_u, \mathbf{l}_v), \forall(u, v)\}$, we can consequently track the consistency of data flow. Beware that considering the cross-correlation of all pairs of points implies working with huge traces—-we consequently opted to somewhat optimise the cross-correlation traces by only storing for each $\mathbf{l}_u$ the maximum of $\{xr(\mathbf{l}_u, \mathbf{l}_v)\}$ over all $\mathbf{l}_v$ that are not 'too close' to $\mathbf{l}_u$, i.e. $\mathbf{xt} = max_v\{xr(\mathbf{l}_u, \mathbf{l}_v), \forall(u, v)\}$ with v not too close to u. We do not allow u and v to be too close because, as explained before, neighbouring points are intrinsically highly correlated and so the cross correlation, whilst being close to one, nevertheless reveals nothing about the data dependency. Consequently our xt traces have the same length of the original traces, and exclude pairs of points with intrinsically (and data independent) high cross correlation.

We demonstrate this in Fig. 4, where we have chosen two devices with contrasting architectures to demonstrate that cross-correlation works irrespective of the underlying device.

The first device features a highly serial implementation of AES where each step only touches at most one byte of the state. It is representative of AES implementations in the low cost market. The second device features a highly parallel implementation of AES. Such an implementation is more likely to be found in high end products where implementation speed and security is considered an important factor.

3.3 Detecting $s\&c$-traces

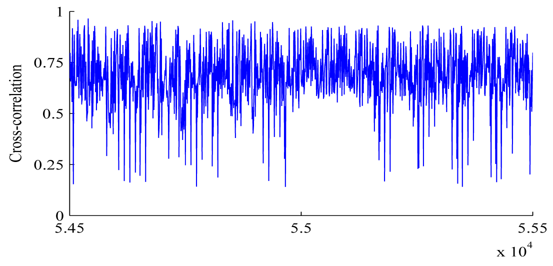
In the proof given in [19], an adversary plays the $q-$sim game and so has access to the oracles Enc, $Leak$, and Gen (see Sect. 2.1). In our distinguisher implementation we do not require access to the oracles $Leak$ and Gen.

To detect the $s\&c$-traces we apply the cross-correlation method to a set of power traces. Recall that because neighbouring points are always highly correlated [11, Ch. 4] we improve our practical implementation for the method by setting a window of a fixed width around each point for which no cross-correlation is computed. The detection is then based on the absence of cross-correlation that would otherwise be present in traces that have not been simulated, i.e. a (set of) points which leak on the same data no longer show a significant correlation with respect to each other. The $q-\text{sim}$ (and $p-q-\text{sim}$) games define that there exists a simulator that is secure for all adversaries (with set computational limits). We may hence assume this includes all implementation details of the PRF as well as the principle of the simulator, i.e. where the traces are split and concatenated. Recall that a cross-correlation trace shows 'patterns' which are based on the relationships between following and related intermediate values. Consequently, even a cross-correlation trace from the simulated leakages will show such patterns, see right hand side of Fig. 4. Only at the round where the simulated traces have been split and concatenated a different pattern will occur. Consequently an attacker gets a 'fingerprint' for how the cross-correlation trace should look (in the case of the *s&c*-simulator) by examining the beginning and end of the cross-correlation trace. Any significant deviation from the fingerprint (i.e. any discrepancy between the two) will hence identify the *s&c*-traces.

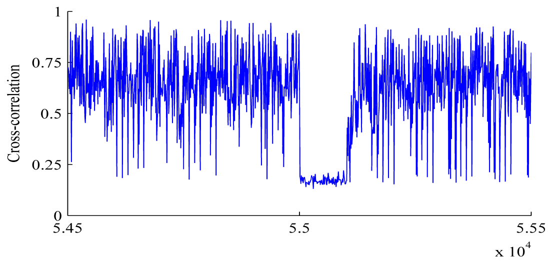
Experiments for real devices. Practical attacks are often played down because they are device specific and hence it is often not possible draw general conclusions from a single attack. For AES there are several implementation options. Serial implementations are often found on small processors (8-bit or 32-bit). These are software-only implementations and we have used two different widely-used processors to instantiate such a software implementation for our attacks. Parallel implementations can be found as dedicated hardware implementations. In practice 32-bit implementations would be considered as suitable for constrained devices such as smart cards, whereas highly parallel 128-bit architectures would be used when throughput is a practical concern. We opted to use a highly parallel 128-bit architecture to provide contrasting results to the software implementations. In the following, we give a brief overview of the results obtained from each architecture. Details regarding the acquisition setup and target devices can be found in Appendix B.

Software implementations. We used a general purpose microcontroller which features an 8051 instruction set as the first device for our attacks. The cross-correlation plots for this architecture reveal detailed information about the data flow during the execution of an algorithm. In our running example (AES-128), we are able to detect the order in which state bytes and functions are touched, the operations performed and for a masked implementation, when and where each masked is applied. In Fig. 5(a) and 5(b) we show a portion of the cross-correlation for real and simulated leakage traces respectively. There is a clear break in the cross-correlation as a result of the concatenation between traces with inconsistent states. We repeated the *s&c* experiment with an ARM based 32-bit architecture and implementation. Like for the 8051, we can track the AES data flow and so the simulated leakage trace, once again, leads to a drop in the cross-correlation. We show the plots for the results of the ARM in Appendix C, Fig. 11.

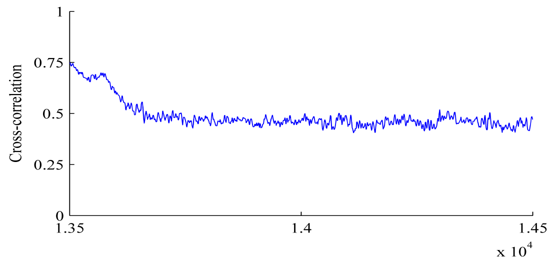
Hardware implementation. The SASEBO-R ASIC boasts a large number of cryptographic functions implemented as dedicated logic. Unlike the two previous devices, the information leaked is no longer dependant on processor operations but on combinatorial switching. Figs. 4(c) and 2(b) show the power consumption over an execution of AES and a simulated trace generated by the *s&c* simulator respectively. The cross-correlation distinguisher now plays on the coherency of the combinatorial switching rather than the state leakage at each clock cycle. As with the software implementations, we can easily identify the simulated leakage traces, see Fig. 5(c) and 5(d). One key difference over the software implementation is cross-correlation no longer reveals any information about the data flow or what operations are being performed but simply that there exists some data dependency in the power consumption of the device.



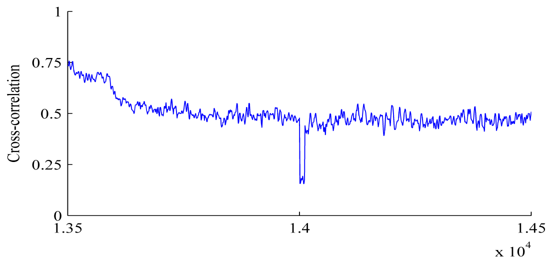
(a) Real trace (8051).



(b) Simulated trace (8051).



(c) Real trace (SASEBO-R).



(d) Simulated trace (SASEBO-R).

Fig. 5. Cross-correlation distinguisher plots.

3.4 Factors that influence the success probability of winning the $q-$ sim game

Remember that we estimate the cross-correlation between trace points, which implies varying inputs and keys. We may however chose to vary the keys k, and use e.g. a single plaintext x for each key. In the $q-$sim game the parameter q represents the number of inputs per key. In the $p-q-$sim game the parameter p refers to the number of different keys k. Consequently, our cross correlation method efficiently hybridises i.e. $\mathbf{Adv}^{p-q-\text{sim}}_{L,S^L,\mathbf{BC}}(A) \leq p \cdot \mathbf{Adv}^{q-\text{sim}}_{L,S^L,\mathbf{BC}}(A)$. A DPA style distinguisher could not take advantage of this and the advantage of the $p-q-$sim game would be equal to the advantage of the $q-$sim game.

Impact of noise on the cross-correlation distinguisher. Just as we can write down a formula for the impact of the signal-to-noise ration (SNR) on a correlation DPA attack (see, e.g. [10, Ch. 4.3.1]), we can express the impact of the SNR on our proposed cross-correlation attack.

We can write leakage points as a direct sum of an (unknown) signal plus independent (Gaussian) noise, i.e. $\mathbf{l}_u = \mathbf{S}_u + \mathbf{N}_u$, with $\mathbf{N}_u \sim \mathcal{N}(0, \sigma_u)$, and $\mathbf{l}_v = \mathbf{S}_v + \mathbf{N}_v$, with $\mathbf{N}_v \sim \mathcal{N}(0, \sigma_v)$ [4,11]. The signal to noise ratio (SNR) is defined as $SNR = \frac{Var(\mathbf{S})}{Var(\mathbf{N})}$. The respective SNRs at the points u and v are then $SNR_u = Var(\mathbf{S}_u)/Var(\mathbf{N}_u)$ and $SNR_v = Var(\mathbf{S}_v)/Var(\mathbf{N}_v)$. Using the fact that we defined signal and noise as independent components contributing to the overall leakage at a point, we can write out how the correlation between two trace points depends on the SNRs at those points:

$$\rho(\mathbf{l}_u, \mathbf{l}_v) = \rho(\mathbf{S}_u, \mathbf{S}_v) \cdot \frac{Var(\mathbf{S}_u) \cdot Var(\mathbf{S}_v)}{\sqrt{(1 + \frac{1}{SNR_u})(1 + \frac{1}{SNR_v})}}$$

The main factor that makes the game hard to win is hence not the 'size' of the intermediate value (in terms of the bit length) itself but the SNRs and the correlation between the (noise free) leakages $\rho(\mathbf{S}_u, \mathbf{S}_v)$. At first glance this goes against the intuition from DPA style attacks where practice has shown that they become harder for architectures that employ a larger data-path. However, this is in part because a predictable portion of the state is small in comparison to the entire state. For instance, attacks on 32-bit processors often only predict 8 bits of the 32-bit state, so the remaining 24 bits are noise. In addition, in DPA style attacks using correlation one requires to 'model' the leakage behaviour, and especially for dedicated hardware, standard models such as the Hamming weight or Hamming distance are less than ideal approximations. The cross-correlation distinguisher however does not require to model the device leakage and, importantly, we are effectively using the entire state information because we are working with points and do not have to make any predictions. This explains the highly effective nature of the distinguisher.

Factors NOT influencing the success probability. It would be tempting to assume that any countermeasure against correlation-based DPA attacks would automatically work against the cross-correlation distinguisher because both distinguishers share the same statistical method.

The two main engineering approaches to distinguish classes of countermeasures are hiding and masking [11]. Hiding countermeasures typically change the SNR and so increase the number of leakage traces that are needed for a successful attack.

Masking (i.e. secret sharing) countermeasures aim at making it impossible to exploit the information by distributing it over different intermediate values (and hence leakage points), such that it becomes increasingly infeasible to 'recombine' that information. In practice however it is not possible to implement secret sharing with many shares. Typically only two, or at most three, shares are used and the masks (i.e. randomness) are not refreshed in between rounds or in between invocations of an intermediate value [8,7]. Consequently, practical masking schemes maintain the consistency between the subsequent transformations on the state and so the cross-correlation distinguisher remains applicable.

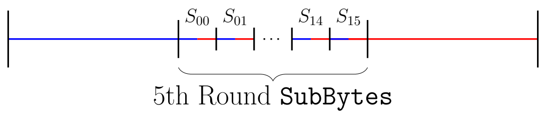
4 Making secure simulators

We now discuss our attempts to make secure simulators by focusing on the areas that we identified in the previous section as contributing to the security of the $q-\text{sim}$ game. The first factor was the SNR, which can be changed for dedicated hardware implementations. However, this will not change the advantage of the adversary in the $p-q-\text{sim}$ game. The second factor was the intrinsic cross-correlation between points. Rather than cutting at 'uninformative' points, it seems more fruitful to cut at points where the intrinsic cross-correlation is low.

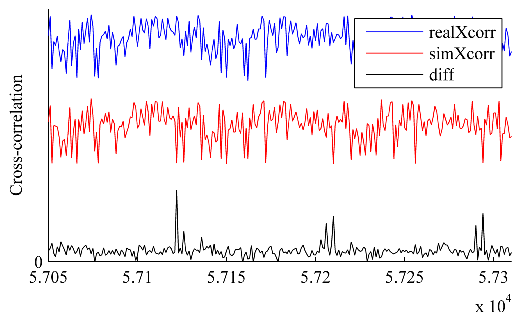
4.1 Cutting at points with low intrinsic cross-correlation

Our first strategy aims to select points related to intermediate values which are *per se* highly uncorrelated. Given our running example is AES, the natural candidate intermediate value is SubBytes, i.e. the input and output of SubBytes are (almost) statistically uncorrelated, hence we can expect that there is also only a low correlation between the corresponding trace points.

Whilst this seems like a compelling idea, it comes with a caveat. SubBytes (like other non-linear functions typically found in block ciphers) can be implemented in two ways. Either a table is stored and hence a SubBytes computation corresponds to a table lookup operation. Alternatively it can be computed as an affine transormation. The former approach is suitable for somewhat 'serial' implementations, as typically only a single instance of the table is held in logic. On the other hand, dedicated hardware platforms favour a combinational circuit transformation as a means to minimize the area constraints over having a copy of the same table per lookup.



(a) A visual illustration for S_{sbc}.



(b) Cross-correlation comparison for S_{sbc} (8051).

Fig. 6. Illustration for a serial S_{sbc} simulator and a cross-correlation comparison of S_{sbc} for the 8051 microcontroller.

We hence consider both implementations where the SubBytes function is implemented as a table lookup for a general purpose microcontroller and a combinatorial circuit for dedicated hardware. We construct a simulator S_{sbc} which 'tweaks' the $S^{L}_{s\&c}$ simulator construction to perform the split-and-concatenate at the S-Box lookup rather than points of 'no activity'. However, we note that for a sequential lookup, the simulator is required to perform splice at each S-Box rather than a simple split-and-concatenate as illustrated by Fig. 6(a).

A practical attempt for an 8051 processor. This method was first evaluated on an 8051 emulator which produces noise-free leakage on the data processed at each instruction. The simulated traces were indistinguishable from real traces produced by the S_{sbc} simulator. We hence proceeded to implement this for a real 8051 device where we were met with the challenges of real world (imperfect) leakage. Although we could pinpoint the exact location of the SubBytes operation, it so happens that each low-level operation is performed over multiple clock cycles and hence leaks multiple times for each operand. To be precise: consider the lookup $r_0 = M[A]$, where a register r_0 is loaded with the contents of memory address A. The resulting leakage trace resembles the form $[\mathcal{L}(A)||\mathcal{L}(M[A])||\mathcal{L}(A)||\mathcal{L}(M[A])]$ for some leakage function $\mathcal{L}$ in our 8051 processor.

As a result, the simulator would need to splice multiple times within each S-Box lookup. This behaviour is very architecture specific and may not necessarily translate to the same result for a different device. Figure 6(b) shows the cross-correlation resulting from splicing at each S-Box; the top plot (printed in blue) shows the cross-correlation trace as derived from real traces. The middle plot (printed in red) shows the cross-correlation trace as derived from simulated (S_{sbc}) traces. Whilst a visual detection seems difficult at first, an adversary with information about the time points (or a fingerprint, which we explained previously can be constructed even from simulated traces) can spot the difference. This is made clear by the lowest plot (printed in black) that shows that there is a distinct difference between real and simulated cross-correlation.

A practical attempt for the SASEBO-R. We now consider the implications of a parallel combinatorial SubBytes function as performed by the SASEBO-R ASIC. Pinpointing the SubBytes operation is no longer such a trivial task as each round function is evaluated as a combinatorial circuit rather than being governed by an external clock. Attempting to model the leakage in an ideal setting would also require significant knowledge of both the design and layout of the ASIC. We hence resorted to a exhaustive search over a whole encryption round in order to determine whether or not a point existed that would allow us to build the S_{sbc} simulator for such a device.

```
**simulator** $S^L(x, c)$:
Perform a meet–in–the–middle attack to learn a valid $(k_1, k_2)$
$\mathbf{BC}_{k_2}(\mathbf{BC}_{k_1}(x)) \rightsquigarrow \Lambda$
Return $\Lambda$
```

Fig. 7. A generic simulator S secure against the cross-correlation distinguisher.

Perhaps unsurprisingly, we were unable to identify points that did not produce a significant drop in the cross-correlation. This is primarily due to the relation between evaluation stages of a combinatorial circuit. Without further insight on the design of the ASIC, we were unable to determine what processes were taking place to prevent us from building a viable S_{sbc} simulator. However, we also make no claim that this is not possible but rather that our attempts were not successful in finding a method to do so.

4.2 Attempting to achieve state consistency

Over the execution of any algorithm there exists a degree of consistency between the intermediate values, which is disrupted by splitting traces. Hence, we attempted to design a 'state aware' simulator. Our first attempt consisted of generating an 'intermediate round' that ensures such consistency. However, our attempt, which we show in App.A, requires to select an appropriate key for this intermediate round. This key is inconsistent with the other round keys, and this can be detected via cross correlation.

Doubling the cipher. All our previous attempts exploited specifics of AES to build a secure simulator. However an approach based on using a double block cipher (i.e. a block cipher **2BC** that consists of two sequential computations of a block cipher **BC** with independent keys k_1 and k_2) avoids this constraint albeit by placing a heavy computational burden on the simulator (see Fig. 7).

This simulator is performing a meet–in–the–middle attack [10] to derive a pair of keys (k_1, k_2) that encrypts x to c. In the case of AES, this simulator requires approximately 2^{65} AES encryptions per valid trace, which is of course extremely computationally expensive. The simulator maintains state consistency throughout and so the traces it produces cannot be detected via cross-correlation analysis. However, in the face of a standard DPA attack, one would notice that no key hypothesis ever achieves a good correlation with the simulated traces. The best distinguisher for this simulator would hence be a DPA attack. This fact also implies that we cannot play the $q-$sim game 'across' different keys anymore. Each new round of the game (with a new key) poses a fresh challenge for the adversary. Consequently, and as we explained before, the $p-q-$sim game does no longer hybridise over p, and hence the success probability for an adversary is bounded via the interplay between SNR (the signal here depends on the size of the predicted intermediate state which is different to the signal for the cross-correlation distinguisher) and q. Consequently, for a small q and a low SNR a real world attacker will not be able to distinguish between simulated or real traces.

For completeness we show that this simulator can be plugged into the PRG construction from [19] maintaining the correctness of the proof. We only switch out the underlying PRF from AES to double AES, and subsequently we need to switch the $2PRF$ for a $3PRF$ for the extra rekeying material. The proof given in [19] can be expanded for any constant number of calls to the PRF and thus the construction will not need reproving secure. The resulting construction can be seen in Fig. 8.

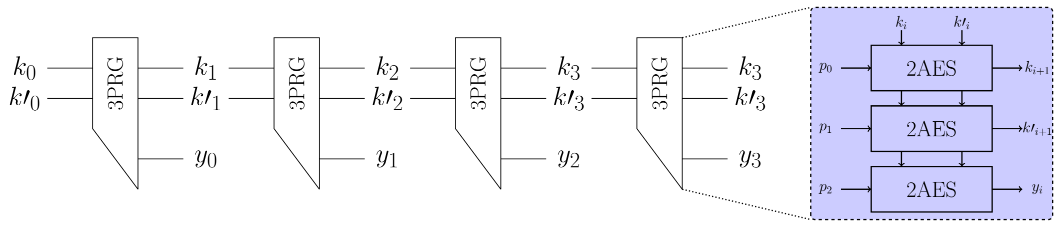


Fig. 8. The adjusted PRG construction.

Acknowledgements

Dan Martin and Elisabeth Oswald have been supported in part by EPSRC via grant EP/I005226/1. Jake Longo Galea has been supported in part by a studentship under the EPSRC Doctoral Training Partnership (DTP) scheme.

References

1. Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 36–54. Springer, 2009.
2. Atmel. AT89S8253 Datasheet. `http://www.atmel.com/Images/doc3286.pdf`.
3. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In M Joye and J-J Quisquater, editors, *CHES*, volume 3156 of *LNCS*, pages 135–152. Springer Berlin / Heidelberg, 2004.
4. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.
5. Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.
6. Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, volume 6110 of *LNCS*, pages 135–156. Springer, 2010.
7. Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *SAC 2010*, volume 6544 of *LNCS*, pages 262–280. Springer, 2011.
8. Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In *ACNS*, volume 3989, pages 239–252. Springer, 2006.
9. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Proceedings of CRYPTO 1999*, pages 388–397. Springer-Verlag, 1999.
10. Xuejia Lai and James L. Massey. Hash function based on block ciphers. In Rainer A. Rueppel, editor, *EUROCRYPT*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 1992.
11. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Advances in information security. Springer, 2008.
12. Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.
13. Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In *CHES*, pages 144–157. Springer, 1999.
14. Silvio Micali and Leonid Reyzin. Physically observable cryptography. In *TCC 2004*, LNCS 2951, pages 278–296. Springer, 2004.
15. NXP. LPC2124 Datasheet. `http://www.keil.com/dd/docs/datashts/philips/lpc2114_2124.pdf`.
16. SASEBO. SASEBO Crypto LSI Specification. `http://www.rcis.aist.go.jp/files/special/SASEBO/CryptoLSI-ja/CryptoLSI2_Spec_Ver1.0_English.pdf`.
17. SASEBO. SASEBO-R Specification. `http://www.rcis.aist.go.jp/files/special/SASEBO/SASEBO-R-ja/SASEBO-R_Spec_Ver1.0_English.pdf`.
18. Laurent Sauvage, Sylvain Guilley, Florent Flament, Jean-Luc Danger, and Yves Mathieu. Blind cartography for side channel attacks: Cross-correlation cartography. *Int. J. Reconfig. Comp.*, 2012, 2012.

19. François-Xavier Standaert, Olivier Pereira, and Yu Yu. Leakage-resilient symmetric cryptography under empirically verifiable assumptions. In *CRYPTO*, volume 8042, pages 335–352. Springer-Verlag, 2013.
20. IAIK TU. DPA Demo Board. `https://www.iaik.tugraz.at/content/research/implementation_attacks/impa_lab_infrastructure/`.
21. Marc F. Witteman, Jasper G. J. van Woudenberg, and Federico Menarini. Defeating rsa multiply-always and message blinding countermeasures. In *CT-RSA*, volume 6558 of *LNCS*, pages 77–88. Springer, 2011.
22. Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger. An asic implementation of the aes sboxes. In *CT-RSA*, volume 2271 of *LNCS*, pages 67–78. Springer, 2002.

```
simulator $S^L_{2c}(k^*, x, c)$:
$c', ST_5 \leftarrow \mathbf{AES}_{k^*}(x) \rightsquigarrow l^{1,4}||\alpha$
$x' \leftarrow \mathbf{AES}^{-1}_{k^*}(c)$
$c, ST_6 \leftarrow \mathbf{AES}_{k^*}(x') \rightsquigarrow \beta||l^{6,10}$
Construct $k^{\#}$ using $ST_5, ST_6$
$ST_6 \leftarrow \mathbf{AES1}_{k^{\#}}(ST_5) \rightsquigarrow l^5$
Return $l^{1,4}||l^5||l^{6,10}$
```

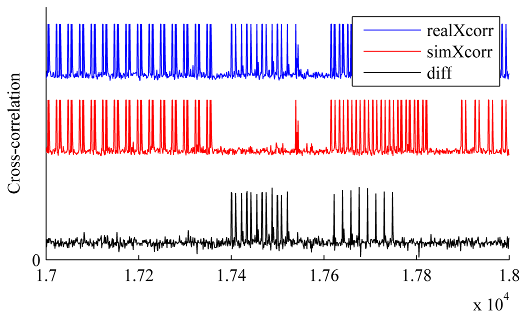


Fig. 9. Simulator description and cross-correlation comparison for S_{2c}.

A Simulator S_{2c}

This simulator shown in Fig. 9 and the extra notation can be understood as follows; ST_i is the state of AES at the start of the i^{th} round and $\mathbf{AES1}_k$ runs a single round of AES on round key k.

The simulator S_{2c} operates similarly to $S_{s\&c}$ by first performing an encryption of x under the key k^*. The leakage captured from this corresponds to the first four rounds $l^{1,4}$. We also store the state ST_5. Next, the encryption of x' under k^* is performed and the leakage of the last 5 rounds is captured $l^{6,10}$ along with ST_6. To connect the two otherwise disconnected states we generate an extra trace $\mathbf{AES1}_{k^{\#}}(ST_5) \rightsquigarrow l^5$. Note that finding the key $k^{\#}$ which maps ST_5 to ST_6 is simple considering only a single round of AES.

We proceeded to implement the S_{2c} simulator for the 8051 emulator and the resulting cross-correlation was once again able to detect the simulated traces. It is perhaps interesting to note that the cross-correlation failed to detect the point at which the traces were concatenated but rather at the inconsistencies now present in the round keys used to construct the fake round. Moreover this further strengthens the notion that cutting at points of 'non-informative points', as in [19], is not always a successful strategy. The reason is that although there might not be any data dependent operations taking place at the point of splitting, the information held before and after are will be data dependent and can ultimately be detected.

B Acquisition Setup and Target Devices

In this appendix we outline the equipment used to gather the side channel data provide a brief note about each of the target devices used throughout the paper.

B.1 Acquisition Setup

The hardware used throughout the experiments follows a typical acquisition setup commonly found in the literature [9] [11, Ch. 3]. The measurement apparatus used for each of the experiments is as follows:

- Tektronix DPO7104 1Ghz Digital Oscilloscope.
- Tektronix P7330 High-performance differential probe.
- TTI EX354 Stable bench power supply.
- Agilent 33220 Signal generator.

The power consumption for each device was captured by measuring the drop across a resistor placed in the ground return path for each of the devices.

B.2 The AT89S8253 8051 Microcontroller

The AT89S8253 [2] is an 8-bit microcontroller which represents the lower end market for hardware. The device can be found in smartcards and is well documented in the side channel community for it's Hamming weight leakage model. This was used in conjunction with the DPA Demo board from IAIK-TU[20].

The AES implementation was limited to an 8-bit serial implementation due to the architectural constraints. Each of the S-Box operations were executed as a table lookup. The device was clocked at 12Mhz throughout all experiments and the oscilloscope set to capture the power signal at 200Ms/s.

B.3 LPC2124 ARM7TDMI NXP Microcontroller

The LPC2124 [15] microcontroller is a 32-bit RISC microcontroller with a 4 stage pipeline. This device serves to represent the mid-range market of microcontrollers with 32-bit architectures. A custom board was designed and used to facilitate power measurement.

The AES implementation consisted primarily of 32-bit operations to build each of the round functions (AddRoundKey, ShiftRows etc.) with the exception of SubBytes which was performed as an 8-bit lookup table. The device was clocked at 14Mhz throughout all experiments and the oscilloscope set to capture the power signal at 250Ms/s.

Additional plots and results from the the ARM 32-bit architecture are shown in Fig. 11.

B.4 SASEBO-R Cryptographic LSI

The SASEBO (Side-channel Attack Standard Evaluation Board) project aimed to provide development kits to facilitate side channel research. The SASEBO-R [17] board is specifically designed to fit a cryptographic LSIs [16]. The AES core used throughout this paper is the AES2 instantiation as illustrated in Fig. 10. Both the clock and power regulation for the target ASIC is managed on the SASEBO-R board. The oscilloscope was set to capture the power signal at 2Gs/s.

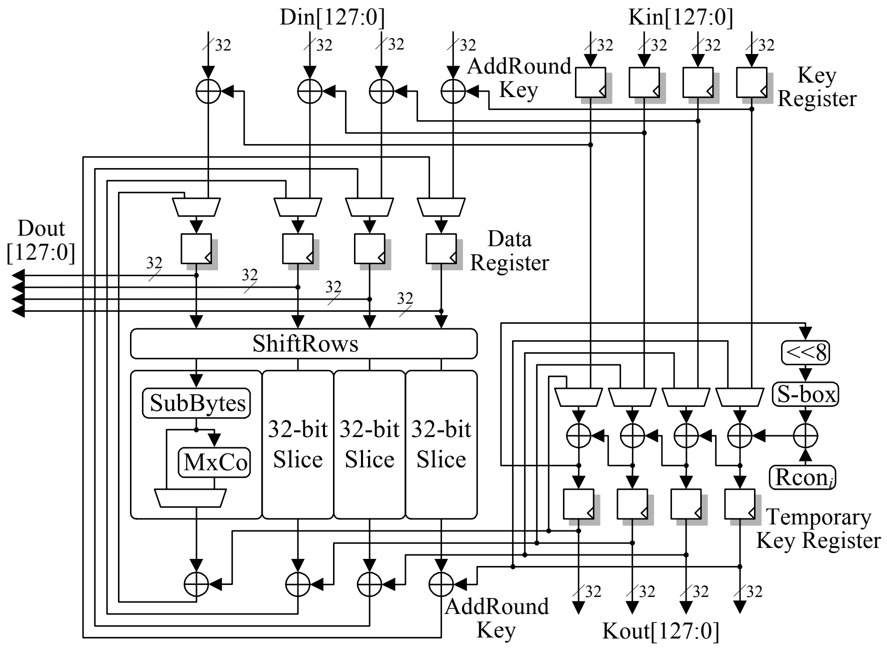
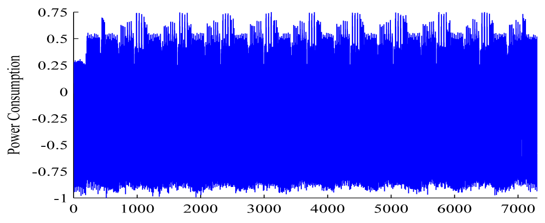
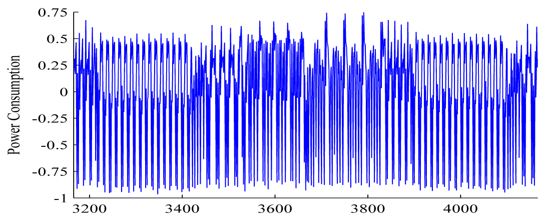


Fig. 10. SASEBO Cryptography LSI AES Encryption Block Diagram [16].

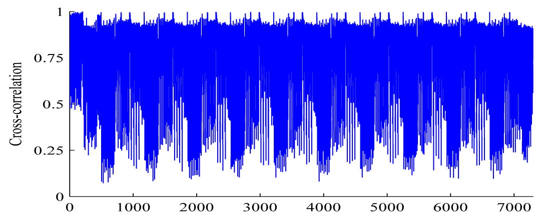
C ARM 32-bit Architecture Plots



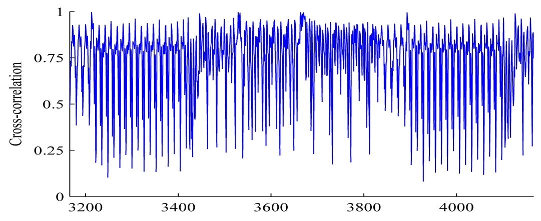
(a) Real AES full encryption trace (ARM7TDMI).



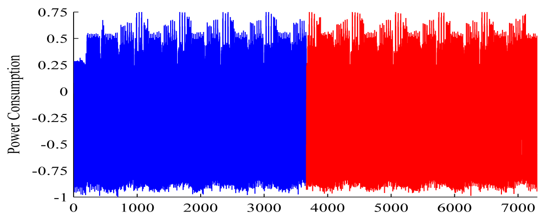
(b) Real AES operations trace (ARM7TDMI).



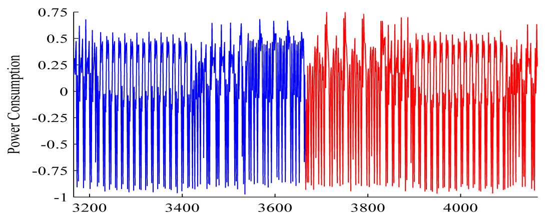
(c) Cross-correlation of a real trace (ARM7TDMI).



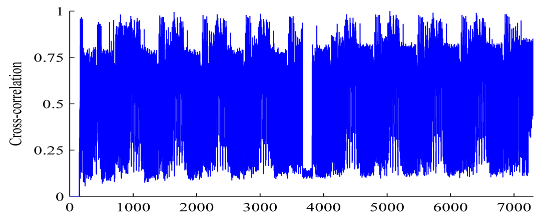
(d) Cross-correlation of a real trace (expanded) (ARM7TDMI).



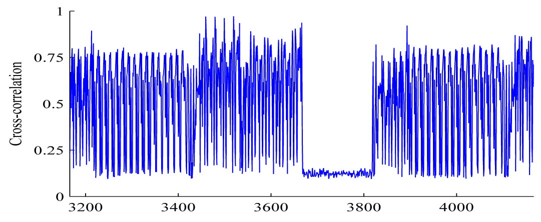
(e) Simulated AES encryption trace (ARM7TDMI).



(f) Simulated AES operations trace (ARM7TDMI).



(g) Cross-correlation of a simulated trace (ARM7TDMI).



(h) Cross-correlation of a simulated trace (expanded) (ARM7TDMI).

Fig. 11. ARM7TDMI encryption and cross-correlation traces generated. Both real and those generated by the $S_{s\&c}$ simulator.