Simulatable Leakage: Analysis, Pitfalls, and new Constructions

J. Longo Galea[1], D. Martin[1], E. Oswald[1], D. Page[1], M. Stam[1], and M. Tunstall[2]

[1] Department of Computer Science, University of Bristol,
Merchant Venturers Building, Woodland Road,
Bristol, BS8 1UB, United Kingdom.
{jake.longo, dan.martin, elisabeth.oswald, daniel.page, martijn.stam}@bris.ac.uk
[2] Cryptography Research Inc.
425 Market Street, 11th Floor
San Francisco, CA 94105, United States
michael.tunstall@cryptography.com

Abstract. In 2013, Standaert *et al.* proposed the notion of simulatable leakage to connect theoretical leakage resilience with the practice of side channel attacks. Their use of simulators, based on physical devices, to support proofs of leakage resilience allows verification of underlying assumptions: the indistinguishability game, involving real vs. simulated leakage, can be 'played' by an evaluator. Using a concrete, block cipher based leakage resilient PRG and high-level simulator definition (based on concatenating two partial leakage traces), they included detailed reasoning why said simulator (for AES-128) resists state-of-the-art side channel attacks.

In this paper, we demonstrate a distinguisher against their simulator and thereby falsify their hypothesis. Our distinguishing technique, which is evaluated using concrete implementations of the Standaert *et al.* simulator on several platforms, is based on 'tracking' consistency (resp. identifying simulator *in*consistencies) in leakage traces by means of cross-correlation. In attempt to rescue the approach, we propose several alternative simulator definitions based on splitting traces at points of low intrinsic cross-correlation. Unfortunately, these 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).

Keywords: leakage resilience, side channel attack, simulatable leakage, cross-correlation

1 Introduction

At Crypto'13, Standaert *et al.* [19] proposed a new notion for leakage resilience involving simulators. The intuition behind their proposal is that if an adversary

cannot tell the difference between real leakage and simulated leakage (from a simulator that does not know the secret key), then clearly the leakage does not reveal any information about the secret key. This offers a middle ground which connects theorists (who desire provably secure scheme) and practitioners (who require empirically verifiable constructions). In this paper we show that while this is a step in the right direction in terms of modelling leakage, the specific simulator given for their construction is in fact distinguishable. We explain why this is the case, and show how to resolve the problem so that their theoretical proof still holds.

1.1 What is leakage?

A 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 three different understandings of what constitutes a leakage function. The first two of these ideas are typically found in works written by practitioners (such as [8,11]) and centre around a mathematical description of the physical nature of real-world leakage traces. The third, 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 is that a leakage function is the mathematical function describing 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.

Leakage understood as modelling of the exploitable information about the key. The second understanding is that of a mathematical function that again describes the leakage traces; however, the conceptual emphasis is that the term 'leakage' refers to leakage about the key. A key must have a finite length and therefore 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 third understanding is that leakage is a function that has certain restrictions (for example, see [4,5]) 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 [15] is an unnecessary expense.

1.2 Simulatable leakage

The recent contribution by Standaert *et al.* [19] hence comes as a welcome addition to the current approaches for 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. 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 therefore 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 c 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 c. This can be done by choosing a random key k^* and computing $c\prime = f_{k^*}(x)$, $x\prime = f_{k^*}^{-1}(c)$, and then $c = f_{k^*}(x\prime)$ 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-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 of two major contributions; first they discuss why state of the art side channel attacks cannot win this game effectively, and, second, they use the game restricted to $q = 2$ to prove a PRG (using AES as the underlying PRF) construction leakage resilient in the standard model.

1.3 Our contribution

We show there exists a side channel distinguisher (against Standaert *et al.*'s simulator when instantiated with AES) which can effectively distinguish simulated traces: it does so by detecting the fact such traces are constructed via split and concatenation 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 we can win the game in an asymptotic setting.

We analyse the properties of (cross-)correlation in this specific application to identify the factors that 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 q-sim game. Our attempts to work with points that have low intrinsic cross-correlation were only successful in theory: for the concrete instantiations in our paper, which are based on AES, there (in theory) should exist such points because of the nature of SubBytes. Theoretically, the input and output of the SubBytes operation are highly uncorrelated. However, 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 instantiating the PRG with a double block cipher construction. Our proposed simulator then uses a meet-in-the-middle technique to determine keys that map x to c without introducing inconsistencies in the data flow. This simulator is somewhat theoretical because of the implied computation cost. However it allows the proof given in [19] to hold once more (remember that this proof crucially depends on the existence of a simulator).

Finally we note in our work that even the computationally expensive simulator still requires noisy traces: it would seem then that the most natural way to produce simulated leakage is to just use the construction 'as is' and 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.

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 $\mathbf{BC}_k(x) \rightsquigarrow l \stackrel{\text{def}}{=} L(k, x)$ 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-128 we can represent a leakage as $l = [l^1, \ldots, l^{10}]$. 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 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., $\mathbf{l}_u = (l_{1,u}, l_{2,u}, ...)$ 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-sim Game

Figure 1 describes the q-simulatable leakage game (q-sim) from [19] for completeness. Recall that the intuition captured in the q-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 $\mathbf{Adv}^{q\text{-sim}}_{L,S^L,\mathbf{BC}}(A) = |\Pr[q\text{-sim}(A, \mathbf{BC}, L, S^L, 1) = 1] - \Pr[q\text{-sim}(A, \mathbf{BC}, L, S^L, 0) = 1]|$.

2.2 Construction

The model given in [19] is used to prove the security of a leakage-resilient PRG which is instantiated using block ciphers (we will continue the pattern started in [19] and will instantiate the block cipher with AES). This construction show on the left in Fig. 2 and the underlying 2PRG is shown on the right in Fig. 2. When considering this within the q-sim game we note that each key is only used twice (once to create the PRG output and one to create the new key) and thus the value of $q = 2$ is the one of interest.

2.3 Simulator

For completeness we also recall the simulator in Fig. 3(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 x under

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

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, Λ)

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 Λ

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

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

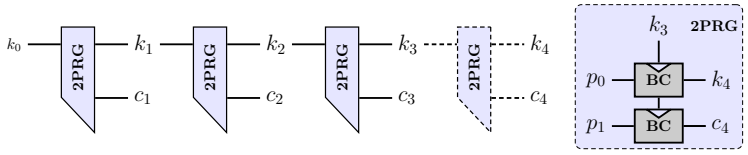


Fig. 2. Left: Leakage resilient PRG. Right: 2PRG construction.

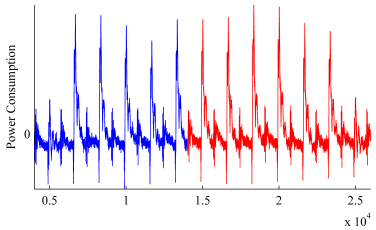
k^* and records the leakage. The next step is to decrypt c under k^* to get a new plaintext x'. The final stage is to encrypt x' under k^* (note that this will encrypt to c) and record the leakage. The leakage is the split (in half) and concatenated such that the first part of the new trace corresponds 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 The Security Game for the Practical Use of the 2PRG: the p-q-sim Game

It is clear that in the q-sim game, the adversary can get only q queries on a key: this represents only a single round in the PRG. However, based on the construction for the 2PRG, we argue that the appropriate security game, which we denote the p-q-sim game, should take into account that potentially many, say p calls to the 2PRG are made. That is, in the p-q-sim game the adversary gets to make q queries against either p real or p simulated instantiations (with

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

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



(b) Simulated $S^L_{s\&c}$ SASEBO-R trace.

Fig. 3. Definition for the $S^L_{s\&c}$ simulator and an exemplary trace.

p different keys) and then he must work out if the leakage he is seeing real or simulated leakage for all of the p instantiations.

We argue further that the p-q-sim game is in general more appropriate: in an evaluation context, which is what [19] really consider, the q-sim game would be played for real, and it seems unlikely that an evaluator would only ever play the game once and take q traces (especially if q is small). More likely, one would play this game several times to get a sense of the success rate for q traces.

It would be tempting to believe that an adversary cannot exploit the information leakage across different games because they are based on different keys. Traditional side channel distinguishers require after all to make key-dependent hypotheses: hence whenever a new key is introduced the attacker is presented with a new, fresh challenge. Whilst [19] discusses why the original q-sim game does not hybridise with respect to q, they do not consider the possibility that the game does hybridise in the number of keys p. If the game were to hybridise, then we would have that $\mathbf{Adv}^{p\text{-}q\text{-sim}}_{L,S^L,\mathbf{BC}}(A) \leq p \cdot \mathbf{Adv}^{q\text{-sim}}_{L,S^L,\mathbf{BC}}(A)$; the game gets easier to win if it is played more often.

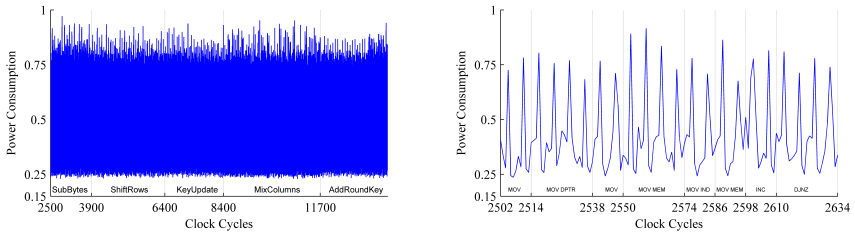
In the next section we will show a distinguisher that can work across different keys and thus can take advantage of the full (and more realistic) p-q-sim game.

4 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-128 as running examples. 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.

4.1 Properties of real world leakage

As argued in previous work [8,3], any implementation of AES will happen as a sequence of steps that correspond to the processing of intermediate values. For instance, in a serial implementation, the state bytes will be accessed sequentially



(a) Power trace showing a single AES round.

(b) Power trace showing individual assembly instructions.

Fig. 4. Power traces for an 8051 8-bit microcontroller.

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. Figure 4 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. 4(a), which represents a single AES round.

In case of this very simple processor, we can even identify the effect (with regard to shape and height) of individual instructions in the power consumption by zooming into the trace further, see Fig. 4(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 operation will touch multiple state bytes but the sequence of round functions must remain sequential. SubBytes is sometimes implemented using combinational logic in dedicated hardware [22]. A combinatorial logic circuit is not governed by a clock but the output from such a circuit will typically be connected to a synchronous storage element. Referring to Fig. 5(c) we can see that there are 10 visible peaks relating to the AES-128 rounds.

4.2 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 [2] 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].

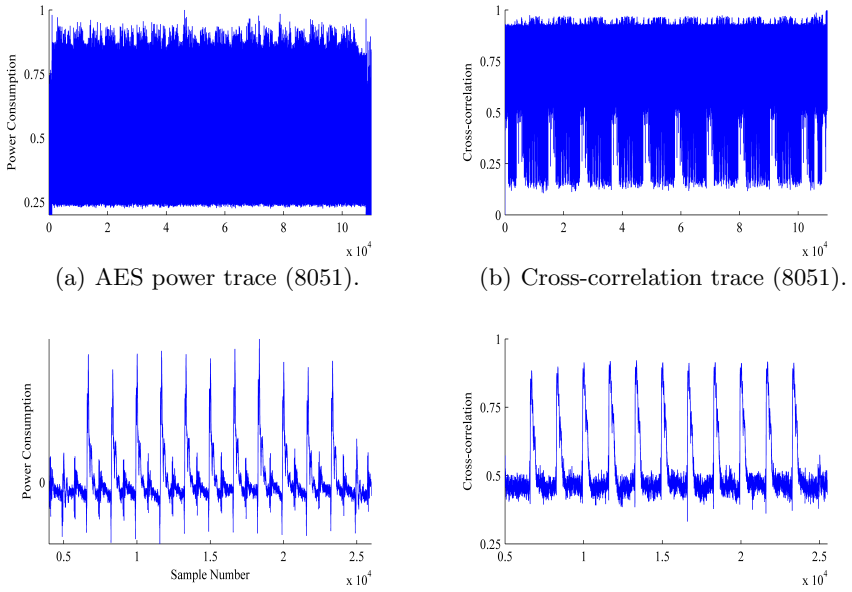
We make the simple observation that the cross-correlation of signals of length one (i.e. points) equates to computing the correlation between trace points $(\mathbf{l}_u, \mathbf{l}_v)$ (as opposed to the correlation with key-dependent predictions in the DPA context). Equation (1) recalls how (Pearson)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

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

Cross-correlation traces We recall that cryptographic algorithms are implemented as step-wise processes (with varying degrees of parallelism). Although AES mixes keying material 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 ShiftRows, 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 (see [11, Ch. 4]), as well as points that are related to the same intermediate values. Further to that we can also expect high 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. $\{\rho(\mathbf{l}_u, \mathbf{l}_v), \forall(u, v)\}$, we can consequently track the consistency of data flow. Such a trace however would be very long (the length would be the square of the original traces' length). We hence opted to reduce the cross-correlation data by selecting the highest cross-correlation value for each u over all 'distant' pairs $(\mathbf{l}_u, \mathbf{l}_v)$, i.e. for each u we took $\widetilde{\rho}_u = max\{\rho(\mathbf{l}_u, \mathbf{l}_v) \forall v\}$ where v is not within a small window around u. Hence our cross-correlation traces $\widetilde{\rho} = \{\widetilde{\rho}_u, \forall u\}$ have the same length as the actual leakage traces, and they have uninformative trivial cross-correlation removed because neighbouring points are not considered. Consequently the cross-correlation traces show the effect of data consistency, and any 'dip' implies some form of discontinuity in the data flow.

We provide in Fig. 5 some cross-correlation traces for illustration. We chose 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



(a) AES power trace (8051).

(b) Cross-correlation trace (8051).

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

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

Fig. 5. AES power traces and cross-correlation plots.

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.

4.3 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* (an adversary may also query them in the p-q-sim game). Our distinguisher does not require any access to the oracles *Leak* and *Gen*.

To detect the *s&c*-traces we apply the cross-correlation method to a set of q leakage traces. 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-sim (and p-q-sim) games define that there exists a simulator that is secure for all adversaries (with set computational limits). We may hence assume the adversary's knowledge 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. 5. 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.

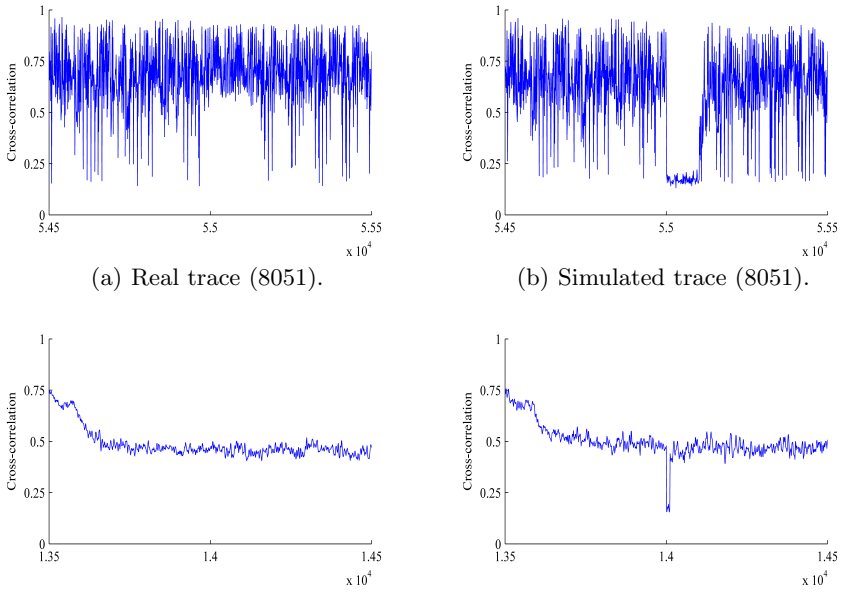
4.4 Experiments for real devices

Practical attacks are often played down because they are device specific and hence it is often not possible to draw general conclusions from a single attack. However, the heart of the proposal in [19] is to be able to implement a secure simulator on some real world devices. We consequently did not want to resort to 'pure simulations' and opted to fully implement simulators for several real world devices. By choosing different devices we can refute the argument that our analysis outcomes are not valid in general: the devices we choose have different architectures that lead to different AES implementations, different leakage models and different noise characteristics.

AES can be implemented in different ways. 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 A.

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 accessed, the operations performed and for a masked implementation, when and where each masked is applied. In Fig. 6(a) and 6(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[1] 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.

[1] The cross-correlation plots for this device are available in the full paper [10].



(a) Real trace (8051). (b) Simulated trace (8051).

(c) Real trace (SASEBO-R). (d) Simulated trace (SASEBO-R).

Fig. 6. Cross-correlation distinguisher plots.

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. Figures 5(c) and 3(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 Figs. 6(c) and 6(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.

4.5 Measures to secure the split-and-concatenate simulator

Recall that we estimate the cross-correlation between trace points, which implies varying inputs and/or keys. The number of different inputs per key is q, whereas the number of different keys is p. We may hence decide to increase p and keep q small, which implies that we can break the PRG construction of [19] because it has multiple rounds. As our distinguisher hybridises, the construction looses security every time a round is leaked, on even though fresh keys may be used!

We now discuss how traditional engineering-style countermeasures impact on the success of winning the game. We begin by showing how noise on leakage traces will help to make winning the game harder. We then explain why masking and hiding approaches are unlikely to defeat the cross-correlation distinguisher.

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

For this we write leakage points as a direct sum of an (unknown) signal ($\mathbf{S}$) plus independent (Gaussian) noise ($\mathbf{N}$), 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 $\mathbf{N}_v \sim \mathcal{N}(0, \sigma_v)$ [3,11]. 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 $SNR_v = Var(\mathbf{S}_v)/Var(\mathbf{N}_v)$. Making the simplifying assumption that $SNR_u \approx SNR_v = SNR$, it turns out (using the same technique as [10, Ch. 4.3.1]) that

$$\rho(\mathbf{l}_u, \mathbf{l}_v) = \rho(\mathbf{S}_u, \mathbf{S}_v) \cdot \frac{1}{1 + \frac{1}{SNR}}$$

In comparison to a DPA attack (we refer to [10, Ch. 6.3]), the impact of the SNR is potentially stronger on this distinguisher. However, because this distinguisher hybridises over different keys, it is also easier to gather more leakage traces to compensate for this. In particular, it follows that asymptotically over an increasing number of keys p we can still win the game for any small q.

More complex/parallel architectures Another important observation at this point is that in contrast to DPA attacks, the 'relative size' of the intermediate value (in terms of its bit length in contrast to the overall device state) itself is less important. 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 (and so the intermediate values that are attacked contribute only a small amount to the overall leakage). For instance, DPA style 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 contrast to typical DPA style attacks, and so the highly effective nature of the distinguisher.

Masking and hiding approaches 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 to make exploiting the information impossible 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 [7,6]. Consequently, practical masking schemes maintain the consistency between the subsequent transformations on the state and so the cross-correlation distinguisher remains applicable.

5 The challenge of making secure simulators

Given that traditional countermeasures are not suitable to rescue the split-and-concatenate simulator, we have to come up with new ideas. Two approaches are (intuitively) worth pursuing. Firstly, this is to try and maintain state consistency across the concatenated traces. Secondly, this is to split where there is a 'natural' discontinuity in the data flow.

5.1 Maintaining 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 by generating an 'intermediate round' that ensures such consistency.

This simulator shown in Fig. 7 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 and the resulting cross-correlation was once again able to detect the simulated traces. This time it detected the discontinuity in the round key schedule. This shows how hard it

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 $l^{1,4}||l^5||l^{6,10}$

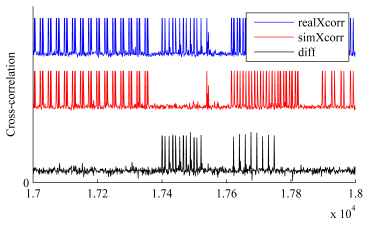


Fig. 7. Simulator description and cross-correlation comparison for S_{2c}.

is to achieve state consistency because we have to take into account the AES state as well as the AES key schedule (in the p-q-sim game), and the fact that different instructions can leak subtly differently.

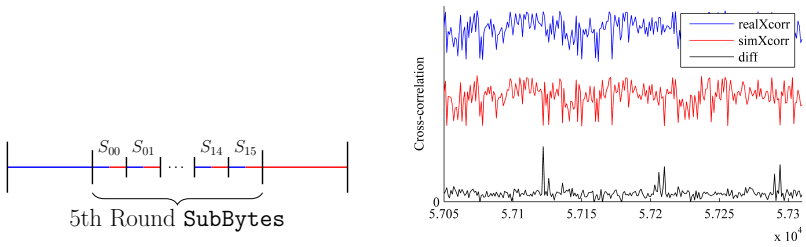
5.2 Leveraging an algorithm-dependent data-flow discontinuity.

Given our running example is AES, the natural candidate intermediate value is SubBytes, because the input and output of SubBytes are (almost) statistically uncorrelated. We can hence expect that there is also only a low correlation between the corresponding trace points, which implies that the data flow across SubBytes is somewhat discontinued. We tested this idea first on an 8051 software emulator which produces noise-free leakage on the data processed at each instruction. The simulated traces were indeed indistinguishable from real traces (taken from the emulator) produced. Consequently we attempted to implement this on real devices.

On real devices, one has two implementation choices for SubBytes. Either a table is stored and hence a SubBytes computation corresponds to a table lookup operation. This is suitable for somewhat 'serial' implementations, as typically only a single instance of the table is held in logic. This is the option used in our AES software implementations. Alternatively, one implements it as combinational circuit, which is hence suitable for dedicated hardware platforms. This option is used in the hardware AES on the SASEBO-R.

Our new simulator S_{sbc} '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. 8(a) (hence we effectively 'chop up' a trace).

A practical attempt for an 8051 processor. Implementing this for a real 8051 device reveals 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



(a) A visual illustration for S_{sbc}. (b) Cross-correlation comparison for S_{sbc}.

Fig. 8. Illustration for a serial S_{sbc} simulator and a cross-correlation comparison of S_{sbc} for the 8051 microcontroller.

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 for the 8051 needs to splice multiple times within each S-box lookup. This behaviour is clearly very architecture specific. Figure 8(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.

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, it is impossible to determine what processes were taking place that made building a viable S_{sbc} simulator impractical.

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

Fig. 9. A generic simulator S secure against the cross-correlation distinguisher.

6 A sound simulator

The previous section has shown that the intuitions for building secure simulators failed to translate to the practical devices that we considered. Furthermore, *how* they failed leaves us with little confidence that using other platforms, e.g. embedded platforms (with other processors) or a different combinational circuit for SubBytes, would be any more successful. The fundamental hurdle is simply that real world leakage is complex, and cryptographic algorithms are necessarily implemented via step-wise processes. Hence there is a specific data flow that will be somehow disrupted when using a split-and-concatenate approach. Without substantial alterations to the devices' designs this cannot be easily changed.

Splitting *within* one instantiation of a block cipher seems hence futile: but how about considering constructions that are based on two somewhat independent block cipher calls?

6.1 Doubling the cipher

Now we discuss 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_i and k_i'): $c = \mathbf{BC}_{k_i}(\mathbf{BC}_{k_i'}(x))$. In this construction there is a natural discontinuity between the first and the second encryption with regards to the key state and so the data flow across the 'boundary' between the first and the second encryption. This makes this boundary an ideal place to 'split' traces, and a generic simulator S that uses a meet–in–the–middle technique [9] to find a suitable pair of keys follows immediately (see Fig. 9).

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 2PRG for a 3PRG 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 to be reproved secure. The resulting construction can be seen in Fig. 10.

6.2 Some final considerations

In the case of AES, this simulator requires approximately 2^{65} AES encryptions (because of the meet–in–the–middle technique) per valid trace. This is computationally too expensive to be practical; yet the simulator is secure against the

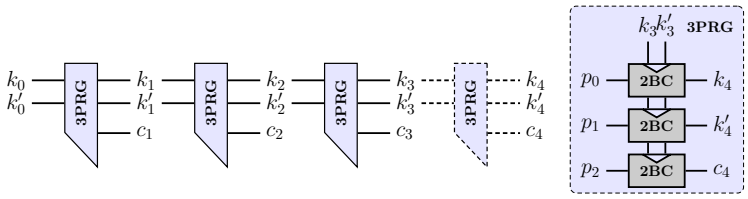


Fig. 10. The adjusted PRG construction.

cross-correlation distinguisher per design so a practical implementation is not necessary in that regard.

However, its security against standard DPA attacks still needs to be considered. Recall that this already is an advantage because DPA attacks do not hybridise over different keys!

Considering now a standard DPA on S^L one would notice (in the process of performing such an attack) that no key hypothesis ever achieves a good correlation with the simulated traces. Hence for a DPA distinguisher we need to consider the question of how many traces are necessary to decide with some certainty that all key candidates are equally likely. We leave this as an option question but note that the usual arguments for DPA success (or lack thereof) will apply. These are that for a (reasonably) low SNR, and a small number of leakage traces q, an attack does not succeed. In particular for the purposes of the PRG construction from [19], which limits q to be two, practical instantiation of S on an ASIC such as the SASEBO-R should be feasible.

Acknowledgements. Daniel Martin and Elisabeth Oswald have been supported in part by EPSRC via grant EP/I005226/1, and Daniel Page by EP/H001689. Jake Longo Galea has been supported in part by a studentship under the EPSRC Doctoral Training Partnership (DTP) scheme.

References

1. Atmel. AT89S8253 Datasheet. `http://www.atmel.com/Images/doc3286.pdf`.
2. E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In *CHES*, pages 135–152. LNCS 3156, 2004.
3. S. Chari, C.S. Jutla, J.R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, pages 398–412. LNCS 1666, 1999.
4. S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.
5. S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156. LNCS 6110, 2010.
6. G. Fumaroli, A. Martinelli, E. Prouff, and M. Rivain. Affine masking against higher-order side channel analysis. In *SAC*, pages 262–280. LNCS 6544, 2010.
7. C. Herbst, E. Oswald, and S. Mangard. An AES smart card implementation resistant to power analysis attacks. In *ACNS*, pages 239–252. LNCS 3989, 2006.

8. P.C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397. LNCS 1666, 1999.
9. X. Lai and J.L. Massey. Hash function based on block ciphers. In *EUROCRYPT*, pages 55–70. LNCS 658, 1992.
10. J. Longo Galea, D. Martin, E. Oswald, D. Page, M. Stam, M. Tunstall. Simulatable leakage: analysis, pitfalls, and new construction. Cryptology ePrint Archive, Report 2014/357 `https://eprint.iacr.org/2014/357`.
11. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2008.
12. S. Mangard, E. Oswald, and F.-X. Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Information Security*, 5(2):100–110, 2011.
13. T.S. Messerges, E. Dabbish, and R.H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In *CHES*, pages 144–157. LNCS 1717, 1999.
14. NXP. LPC2124 Datasheet. `http://www.keil.com/dd/docs/datashts/philips/lpc2114_2124.pdf`.
15. K. Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482. LNCS 5479, 2009.
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. L. Sauvage, S. Guilley, F. Flament, J.-L. Danger, and Y. Mathieu. Blind cartography for side channel attacks: Cross-correlation cartography. *Int. J. Reconfig. Comp.*, 2012(15):1–9, 2012.
19. F.-X. Standaert, O. Pereira, and Y. Yu. Leakage-resilient symmetric cryptography under empirically verifiable assumptions. In *CRYPTO*, pages 335–352. LNCS 8042, 2013.
20. IAIK TU. DPA Demo Board. `https://www.iaik.tugraz.at/content/research/implementation_attacks/impa_lab_infrastructure/`.
21. M. F. Witteman, J.G.J. van Woudenberg, and F. Menarini. Defeating RSA multiply-always and message blinding countermeasures. In *CT-RSA*, pages 77–88. LNCS 6558, 2011.
22. J. Wolkerstorfer, E. Oswald, and M. Lamberger. An ASIC implementation of the AES sboxes. In *CT-RSA*, pages 67–78. LNCS 2271, 2002.

A 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.

A.1 Acquisition Setup

The hardware used throughout the experiments follows a typical acquisition setup commonly found in the literature [8] [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.

A.2 The AT89S8253 8051 Microcontroller

The AT89S8253 [1] 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.

A.3 LPC2124 ARM7TDMI NXP Microcontroller

The LPC2124 [14] 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.

A.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. 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.