Ark of the ECC

An open-source ECDSA power analysis attack on a FPGA based Curve P-256 implementation

Jean-Pierre Thibault[1], Colin O'Flynn[1,2], and Alex Dewar[1]

[1] NewAE Technology Inc, Canada
[2] Dalhousie University, Canada
{jpthibault,coflynn,adewar}@newae.com

Abstract. Power analysis attacks on ECC have been presented since almost the very beginning of DPA itself, even before the standardization of AES. Given that power analysis attacks against AES are well known and have a large body of practical artifacts to demonstrate attacks on both software and hardware implementations, it is surprising that these artifacts are generally lacking for ECC. In this work we begin to remedy this by providing a complete open-source ECDSA attack artifact, based on a high-quality hardware ECDSA core from the CrypTech project. We demonstrate an effective power analysis attack against an FPGA implementation of this core. As many recent secure boot solutions are using ECDSA, efforts into building open-source artifacts to evaluate attacks on ECDSA are highly relevant to ongoing academic and industrial research programs. To demonstrate the value of this evaluation platform, we implement several countermeasures and show that evaluating leakage on hardware is critical to understand the effectiveness of a countermeasure.

Keywords: power analysis · ECDSA · FPGA evaluation

1 Introduction

Side-channel power analysis attacks against cryptographic implementations are well-known in practice, starting with their seminal introduction in 1999 [16]. Since then, a considerable amount of work has been focused on symmetric algorithms, and in particular AES. Power analysis against AES has been demonstrated in real-life examples of software and hardware [21,22,28,18,25,8,32] attacks, and a reader can refer to widely available material such as published books [20], training courses, community driven tutorials such as part of the ChipWhisperer project [24], and open-source implementations of AES power analysis attacks are available as part of numerous open-source projects including OpenSCA [26], ChipWhisperer [24], FOBOS2 [4], Side Channel Marvels [2,7], LASCAR [29] and many others. While side-channel attacks on ECC apply the same basic SPA and DPA techniques, we find that open-source tooling for ECC remains more limited than that of AES.

This may be surprising, as attacks and countermeasures have been discussed since the initial demonstration of power analysis on ECC in 1999 [9]. Many examples of power analysis attacks on ECC are summarized in forward references from [9], along with survey papers on ECC hardware attacks [3,10]. Complete attacks on end-user ECC implementations—for example attacking the standard libraries on mobile phones and computers [6,11] have demonstrated that these attacks are highly practical. Many of these attacks target implementations running on a general-purpose processor, not a hardware accelerator.

As ECC becomes more popular in embedded devices, hardware accelerated ECDSA implementations become increasingly important to evaluate. Work attacking a light-weight ASIC implementation of ECDSA for RFIDs has been shown previously [14]. A complete example of an attack on a "secure" hardware security token with a ECDSA co-processor is given in [19][3]. This security token is based on the NXP A7005 security IC, which claims to have a public-key co-processor; however, details of its implementation are of course unknown.

For research and evaluation purposes, FPGA-based implementations are especially interesting due to our ability to modify them. The results of attacks on FPGAs have been presented previously, but to our knowledge the implementations have not been released in a reusable manner [5,27,31]. These implementations may be designed primarily for side-channel analysis work, and thus are not themselves usable by "end users" as-is.

1.1 Contributions

To understand the side-channel leakage of ECC implementations, we target a high-quality open-source ECC hardware core developed for the CrypTech project, which provides a unique opportunity to study both (a) the viability of well known ECC attacks on hardware implementations and (b) the efficacy of countermeasures against these attacks. While the targeted core claims no explicit side-channel countermeasures, its careful design clearly aims to keep the surface of the secret-dependent logic to a minimum and gives it some intrinsic side-channel attack resistance.

We first show that the target's side-channel leakage is very localized in both time and space. We then show how this small leakage can be exploited in two different ways to get very close to a single-trace attack. The target's limited leakage surface provides the opportunity to reduce the side-channel leakage. We first show how some approaches for mitigating the leakage can fail spectacularly, which highlights that hiding leakage is not easy; we then show two effective implementation countermeasures and compare their efficacy and cost.

In addition to the target being open-source thanks to the CrypTech project, all of the attack code, tools and platform used for this work are also open-source. All power measurements are done using the open-source ChipWhisperer capture tools. The target is implemented on a ChipWhisperer CW305 FPGA target

[3] Note that the attack is on the proprietary 'Titan' security key based on a NXP security IC, which is not related to the OpenTitan project discussed later.

board. Finally, all of the evaluation and attack code is available in the form of Jupyter notebooks, including the example power traces which can be used to recreate this work and all figures in this paper.

We wish for this paper to serve as a useful guide to practitioners in the field, as a reminder that the smallest of leakages can be exploited, and that the adage of trust but verify is critical. By releasing the all of the code associated with this work, we hope to push forward the state of open-source evaluation tooling in assisting future researchers recreating and extending such work.

1.2 Open Source RoT

Most secure boot solutions rely on some Root of Trust (RoT) as part of the boot process, which may also be implemented as a Hardware Security Module (HSM) or Secure Element (SE). Revelations around state-level interference in cryptographic products by Edward Snowden led to a strong interest in open-source hardware implementations, which was part of the push towards the formation of the CrypTech project in 2015 [1].

CrypTech project is the first serious attempt at a fully open-source HSM or RoT device. CrypTech targets a hardware board called the CrypTech Alpha, which combines an FPGA, a microcontroller, and a noise source. In this way, the open-source cryptographic algorithms can be loaded onto a hardware device without requiring a complete ASIC implementation. Validation of the loaded FPGA bitstream can also be performed using standard secure boot processes.

Several other root of trust projects in development will also benefit from this analysis. Ongoing open-source RoT projects with available code include OpenTitan (lowRISC) and Betrusted.io; other projects such as Tropic Square do not yet have public code. The majority of these projects involve an FPGA implementation as part of the development (even if ultimately targeting an ASIC), and thus the work in this paper on the CW305 board is highly reusable by many current projects.

1.3 Paper Structure

The rest of this paper is structured as follows: in Section 2 we provide background on the target implementation and the capture setup. Section 3 walks through the process of identifying the leakage and building successful attacks which can be applied in a real-world scenario. Section 4 studies the cost and efficacy of several countermeasures aimed at mitigating attacks. Finally, Section 5 shows how TVLA testing can be used for detection of leakage on this implementation, but is too blunt a tool for recreating the work done in Section 3.

2 CrypTech and ECC Implementation

The CrypTech project has several elliptic curve algorithm implementations including ECDHP, ECDSA, and EdDSA. In general the cores are designed to work on the CrypTech Alpha board, which contains a Xilinx Artix A200 FPGA alongside a STM32F429 microcontroller and other features (noise source, key memory, etc). The cores have minimal usage of device-specific features, making porting of the cores to other platforms relatively straightforward. The cores from the CrypTech.is project are available at https://trac.cryptech.is.

We have chosen to target the use of ECDSA curve P-256 due to its popularity in current products, and thus make this work and artifact usable by both industrial and academic researchers. The CrypTech accelerator for ECDSA curve P-256 can perform any of five basic operations: equality, copy, modular addition, modular subtraction, and modular multiplication. In addition, the core contains several 'microprograms' that use those basic operations to implement higher-level functions: a curve point doubling, addition of curve point to the base point, and projective Jacobian coordinates to affine coordinate conversion (required by the previous two microprograms). These microprograms run entirely within the core.

In this paper we study the leakage of the point multiplication operation; the objective is to build an attack which retrieves the secret scalar multiplicand k (which allows the private ECDSA key to be trivially calculated). While the target implements a particular curve, the methods developed here should also prove useful for different implementations of different curves.

2.1 Hardware Setup

We have implemented the ECDSA core on a Xilinx Artix A100 FPGA, the FPGA being part of the ChipWhisperer CW305 target board[4], which contains a low-noise power supply and shunt resistor to provide power measurement, and also provides a USB interface for FPGA configuration and communication. The code (ported from the CrypTech project) is available at https://github.com/newaetech/chipwhisperer/tree/develop/hardware/victims/cw305_artixtarget/fpga/cryptosrc/cryptech/ecdsa256-v1. Other cores in the CrypTech project can be similarly ported to our evaluation setup. For power measurement we use a ChipWhisperer CW1200 capture platform (the fully open-source ChipWhisperer-Lite can also be used). The complete setup as used in this paper is shown in Figure 1.

The ChipWhisperer samples the power measurements synchronously to the target device clock [24]; this allows reliable power analysis measurements with low capture rates, and has been shown to outperform much higher asynchronous sampling rates [23]. The target device is clocked at 10 MHz, and power measurements are obtained at 10 MS/s (one sample per target clock cycle). The target is instrumented with a "trigger" output pin which is used to synchronize the collection of power samples with the start of the target operation. Both the target and the capture device are controlled from a single Python program.

To build the attack, the target core is used in ways that would not normally be allowed by an ECDSA core, such as using a known and repeated secret scalar

[4] See https://rtfm.newae.com/Targets/CW305%20Artix%20FPGA/ for details of this board.

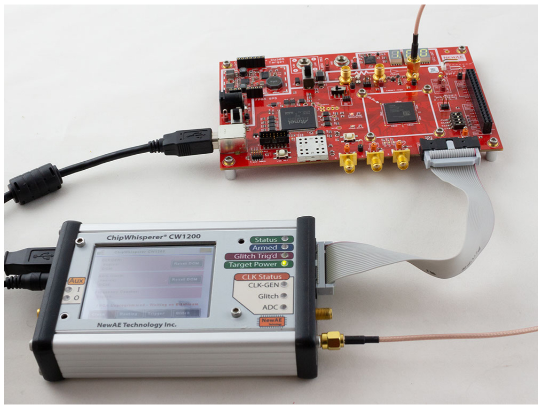

Fig. 1. ChipWhisperer CW305 target and CW1200 measurement tool.

multiplier k, to more easily find and identify the leakage. The final attack requires no such tricks or abuse and can be run successfully on single-use unknown k.

3 Power Analysis of P-256 ECDSA

We focus exclusively on the point multiplication operation. We begin by exploring the target characteristics, and then see how features can be identified to build a distinguisher and complete the ECDSA attack. This attack will primarily use classic "difference of means" (DoM) techniques, but this work can be extended to use other feature selection and distinguisher methods that have shown to be useful with non-profiled ECC attacks [27].

3.1 First Traces: Target Characteristics

The point multiply operation runs in constant-time and always takes exactly 1124157 clock cycles, independent of k or the base point P. Figure 2 shows the first 20000 clock cycles of the point multiply power trace. The regular peaks in this trace shows strong periodic elements to the target operation.

The first step towards building the attack is to identify when each bit of k is processed. This information can be inferred from a power trace or it can be obtained by simulating the Verilog source code and observing the internal workings of the core. We do the latter, since having a simulation waveform will be helpful later for understanding and addressing the leakage. The internal signal `U_curve_mul_256.bit_counter` identifies the bit index of k being processed. We observe that the first bit is always processed 42 cycles after the "go" command is given, and that *each* bit of k takes *exactly* 4204 cycles to process.

This gives us important clues about the implementation: constant-time execution for both the full point multiply operation and each individual bit of k suggests a "double-add-always" algorithm, and that leading zeros are not processed any differently. This constant-time execution is a double-edged sword:

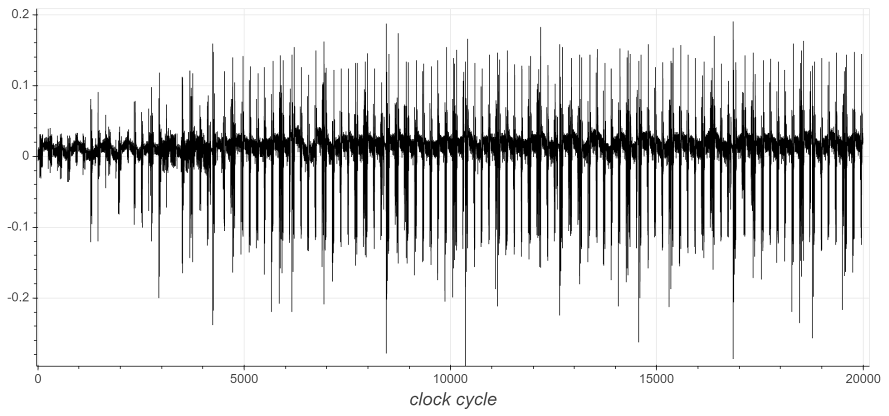

Fig. 2. Raw power trace: beginning of point multiply operation.

while it prevents timing attacks, it also means that once we've learned when the core processes each bit, we don't have to do any further work to accurately find the bit processing times associated with *any* power trace.

Figure 3 overlays the power trace segments for 4 bits of k (a mix of ones and zeros). The peaks line up perfectly. This suggests that differences between $k_i = 1$ and $k_i = 0$ may not be large, and it motivates the next step: we capture a single power trace for a known k and divide the per-bit power trace segments into two bins: one for $k_i = 1$, and one for $k_i = 0$. We then compute the average power trace segment for each bin, and plot the difference between these two average power trace segments.

Figure 4 shows how a power trace for processing $k_i = 1$ differs, on average, from a power trace for processing $k_i = 0$, for each clock cycle of a k_i processing event. We find two significant markers; zooming in, Figure 5 shows that the largest differences occur at cycles 6, 7, 4202, and 4203. These results suggests that we may be able to distinguish between $k_i = 1$ and $k_i = 0$ in the general case using a DoM distinguisher. However we must keep in mind that the preceding figures are obtained by averaging 128 bit processing events; for a practical attack to be successful, individual k_i bits must be distinguished from single power trace segments, without the benefit of averaging.

We can get a first indication of whether the power measurements at these clock cycles can serve as good distinguishers by measuring several power traces with a fixed k (P is variable) and plotting a distinguisher metric for each bit of k. We define a distinguisher metric D as follows:

$$D_i = -p_6^i + p_7^i + p_{4202}^i - p_{4203}^i, i = 0, \ldots, 255 \qquad (1)$$

where p_j^i is the power measurement for the j^{th} clock cycle of the power trace segment for the processing of k_i.

We choose a k to be the concatenation of 128 ones followed by 128 zeros, so that D can provide a clear visual representation of how well the distinguisher

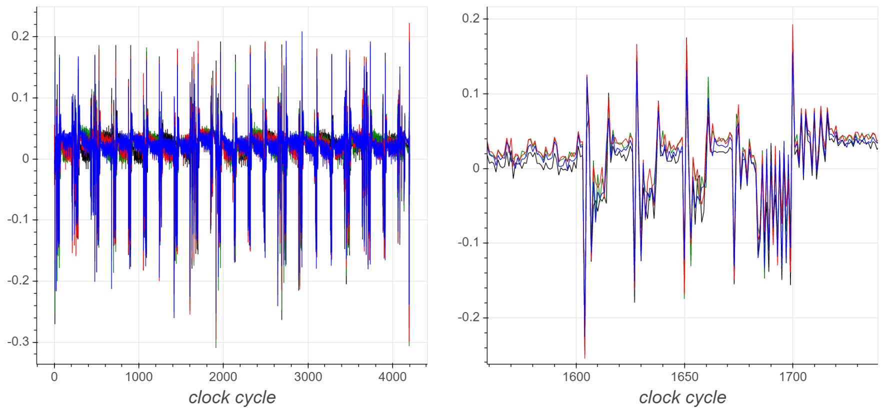

Fig. 3. Power trace segments for 4 bits of k.

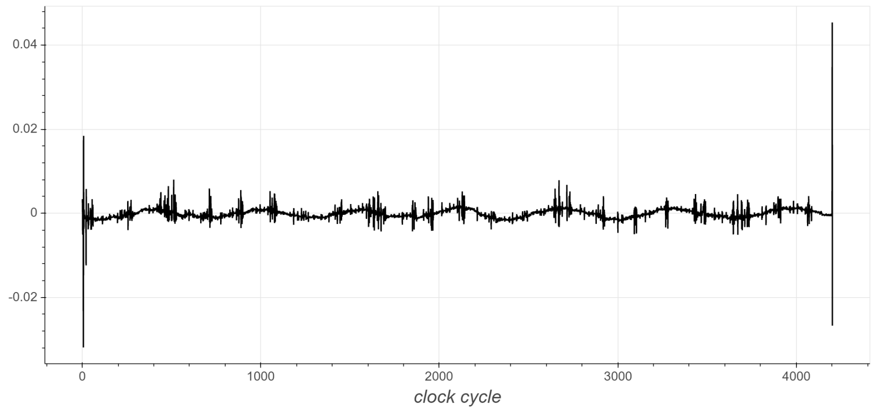

Fig. 4. Difference between the average power trace for $k_i = 1$ and the average power trace for $k_i = 0$.

works. Figure 6 shows D for a single trace on the left, and D averaged over 5 power traces on the right. With a single trace, while there is a clear statistical difference between the ones and zeros of k, several bits would be guessed incorrectly; with 5 averaged traces, all bits of k could be correctly guessed.

3.2 Understanding the Leakage

We next choose a value of k which allows to see whether alternating ones and zeros can be distinguished:

$k =$ `0x0000ffffffffff000000000000ffff00aaaa0000cccc00001111000033330000`

Figure 7 shows that the power measurements at each of the 4 previously identified clock cycles of interest leak the secret k bits roughly equally well, with some difference in how the leading zeros and the first one of k manifest

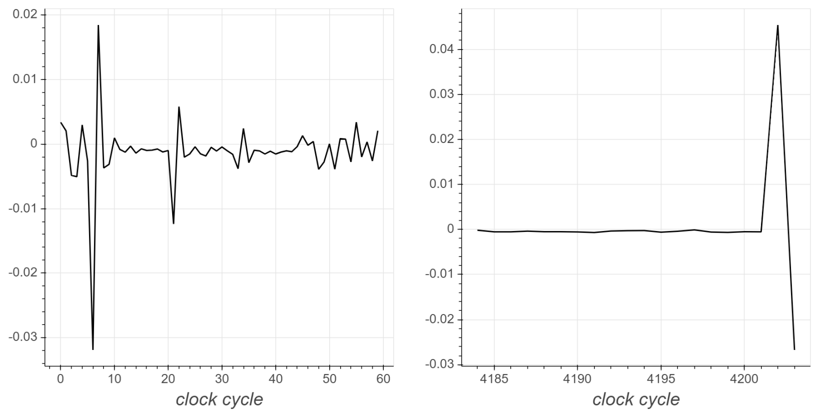

Fig. 5. Zooming in on the significant differences between the average power trace for processing $k_i = 1$ and the average power trace for processing $k_i = 0$.

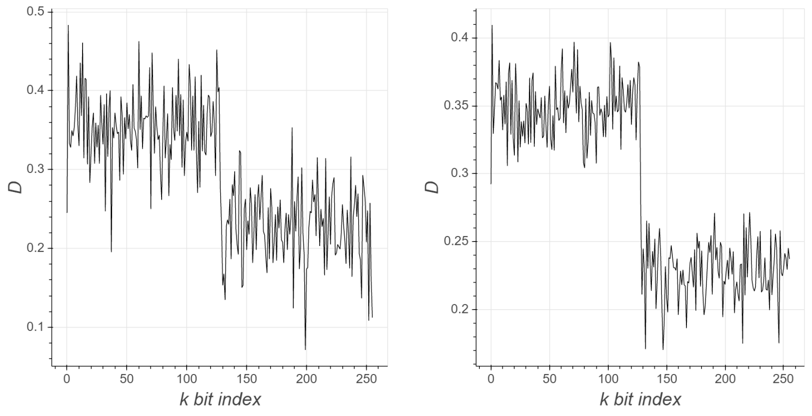

Fig. 6. Distinguisher with one trace (left) and averaged over 5 traces (right).

themselves. Zooming in the area of quickly alternating values, the top two curves (cycles 6 and 7) are offset from the bottom two (cycles 4202 and 4203) by one clock cycle. We therefore adjust the distinguisher metric:

$$D_i = -p_6^{i+1} + p_7^{i+1} + p_{4202}^{i} - p_{4203}^{i}, i = 0, \ldots, 254 \qquad (2)$$

At this point it is helpful and educational to learn *what* the target is doing during these clock cycles. We don't want or need a full understanding of the point multiplication algorithm and its implementation; just enough to gain some insight into why this leakage is occurring, how it may be leveraged for an attack, and how it may be reduced.

Most of the point multiplication control logic is located in `curve_mul_256.v`. Following the k input shows that it uniquely drives a `move_inhibit` signal which is used to mask the write enable control line to a set of three memories: `bram_rx`,

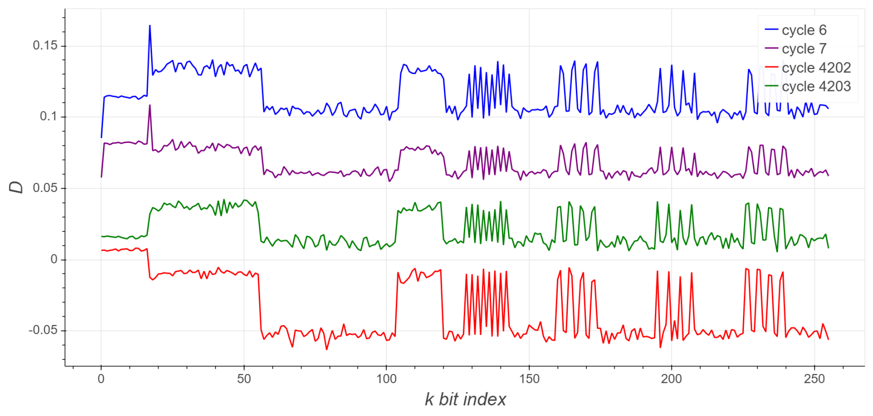

Fig. 7. Distinguisher components for k with sequences of alternating bits, averaged over 50 traces.

`bram_ry` and `bram_rz`. Returning to the earlier simulation of the Verilog source, we observe that `move_inhibit` is only ever active at cycles 4195–4202, which coincides with the power trace differences observed at cycles 4202–4203. These conditional memory writes are occurring at the very end of each k bit's processing time. This aligns with our earlier guess of a double-add-always algorithm; `move_inhibit` is almost certainly being used to selectively keep or discard the result of the intermediate addition which is performed for each bit.

The other leakage was observed at cycles 6 and 7. Simulation shows that the `bram_rz` memory is being read around this time. This is occurring soon after the memory writes that are controlled by k. Moreover, we find that `bram_rz` is written with an incrementing address but is read with a decrementing address, meaning that the first read is for the same 32 bits that were written on the last write; without knowing any details regarding the implementation of Xilinx block RAM (BRAM) memory cells which implement `bram_rz`, it is reasonable to assume that reading what was last written to a BRAM has a power signature which is different from the general case.

We now have a good understanding of the leakage we have observed. We can also use what we have learnt to formulate a slightly different attack approach: instead of using the DoM technique, we could look at the correlation between the power measurements when `bram_r[x|y|z]` are written and the power measurements when each of the three memories are next read. Figure 8 shows these correlations for the same k as that of Figure 7. The SNR for both sets of measurements appears similar, suggesting that either approach may work equally well. The only anomaly with the correlation approach is that correlation on the R_z coordinate appears to only leak the position of the first one.

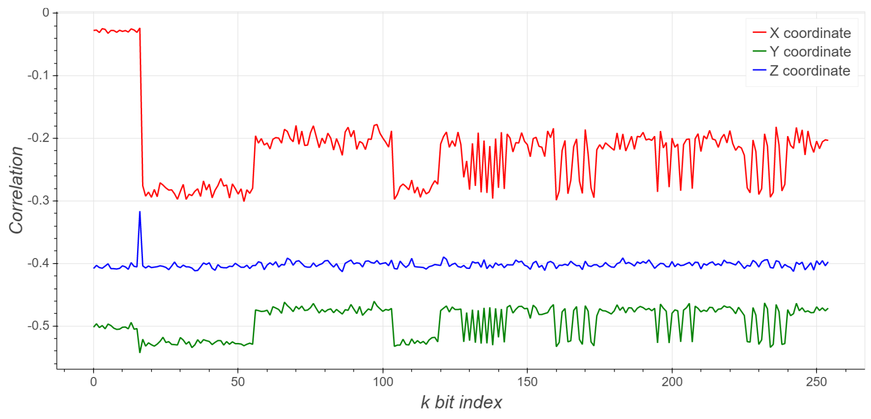

Fig. 8. Using correlation between $R_{x,y,z}$ reads and writes for k with sequences of alternating bits, averaged over 50 traces, vertically shifted for better visual presentation.

3.3 Attack Results

Single-Trace Attack Margin We first examine how many traces are required to fully recover k, where each trace employs the same k but a different base point P. This is not a realistic attack scenario since proper use of ECDSA does not allow repeated use of the same k; the goal of this test is to obtain a measure of the target's margin against a single-trace attack. We synthesize a single power trace by averaging all the power traces. We then use the results of the previous section to guess each bit of k. We repeat this for a decreasing number of averaged traces and find that with the correlation attack, 9 traces must be averaged to correctly guess all the bits, while with the DoM, only 6 traces are required. Table 1 shows the complete results. Since the DoM distinguisher outperforms the correlation distinguisher, we will focus on it in the following sections.

	Number of bad guesses	
Number of averaged traces	Correlation	DoM
9	0	0
8	1	0
7	1	0
6	3	0
5	5	1
4	10	5
3	16	3
2	28	10
1	38	22

Table 1. Number of guesses required.

Hidden Number Problem Carrying out the previous attack with a single trace, we find that on average only 29 bits are incorrectly guessed[5]. Not knowing *which* bits are incorrect would prevent an attacker from going any further with this. However, the distance of the leakage from decision threshold for each k_i can be used as a level of confidence for each k_i and obtain *partial* single-trace guesses for k; if these partial guesses are sufficiently accurate, we can turn this into an instance of the Hidden Number Problem (HNP) and fully recover k. With HNP, we collect several power traces, each for a different k, and for each individual trace we make a partial guess for that k. With enough sufficiently accurate partial guesses, the HNP can be successfully solved to retrieve one of the k used.

HNP requires the known bits of k to be the most or least significant bits of k. This may be a feasible approach here, however this would discard much of the observed leakage. Instead, we turn to the Extended HNP (EHNP) [13], which can make use of known bits anywhere in k, as long as the number of consecutive known bits is sufficient. We follow the approach of [19], which retains traces meeting at least one of the following conditions:

1. at least three runs of 3 consecutive bit guesses;
2. at least two runs of 4 consecutive bit guesses;
3. at least one run of 5 or more consecutive bit guesses

To obtain an approximate measure of the feasibility of solving the EHNP, we heuristically adjust the threshold for accepting guesses such that, out of the traces which are identified as having enough consecutive guesses (as per the criteria above), all but one trace is completely free of incorrect guesses. We collect 10000 individual traces (each with a different random k) and find that we can make a partial k guess which satisfies the consecutive criteria for 7 traces (of which 1 trace has one or more bit guesses which are actually wrong). In the next section, we study countermeasures aimed at improving this figure.

4 Implementation Issues and Countermeasures

Countermeasures for software ECC implementations are well-known and numerous. Many are not directly applicable to hardware implementations, as the source of leakage can differ. Previous work on hardware countermeasures in [5] and [31] is more relevant to our work. Some of these are for different curves; others make different assumptions on the implementation making direct application impossible (short of a complete redesign). Instead, we turn our attention to what has been termed "implementation leakage" [10]. Since we have some understanding of the origin of the leakage which enables our attack, are there modifications to the existing implementation which can effectively mitigate our attack? This approach can be relevant when improvements in side-channel resistance are required and a redesign is not practical.

[5] In the previous section we were averaging *traces*; here we are averaging the success rate of multiple *single-trace* attacks.

4.1 Naïve Targeted Approach (Attempt 1)

Since the leakage that was exploited in the previous sections originates from the conditional writing of intermediate results to memory, a naïve approach at hiding this leakage is to somehow always write the intermediate results. This can be done by doubling the memory space so that in the case where the intermediate results should not be written, they can now be written to the newly created "dummy" memory space.

Hardware synthesis tools are as sneaky as software compilers in their attempts to eliminate functionally unnecessary logic, so care must be taken to prevent the compiler from recognizing the expanded memory space as being not functionally required. To ensure that the increased memory space has the same properties as the original memory, we force the expanded target memories to be implemented as Xilinx BRAM instances, just as they are in the original design. This helps ensure that the increased storage is not implemented in a way that leads to wildly different power consumption signatures.

Unfortunately, not only is this not an effective countermeasure, it actually *increases* the leakage. There are two things to consider for evaluating the countermeasure's utility: (1) whether the leakage is harder to find; (2) whether the attacks are harder to execute.

Is the Leakage Harder to Find? Figure 9 shows that the leakage markers get slightly *stronger* with the countermeasure, which underscores how hard it can be to "hide" leakage in practice. The leakage at cycles 6–7 is easiest to understand: recall that we hypothesized that this leakage is due to reading the same data that was last written to the memory; this hasn't changed here. As for leakage at cycles 4202–4203, our original hypothesis was that this leakage was due to the act of writing (versus not writing) the target memory. These results suggest that the leakage originates from the *control logic* for the writes, rather than the writes themselves. The countermeasure did not eliminate the secret-dependent write control logic; it merely altered it.

Are the Attacks Harder? Using the DoM metric, we find that guessing k from a single power trace now gives on average 21 wrong bits per trace, compared to 29 for the original target. When using multiple traces, the correct k is guessed with 4 traces, compared to 6 for the original target. The correlation leakage also remains. With the original target, we were computing the correlation between (a) writing and reading the same data from memory, and (b) no write activity and reading data from memory. Now, (a) remains unchanged, while (b) has changed to writing "some data" and reading "different data" from memory. Intuitively, we should not expect the delta between (a) and (b) to decrease.

4.2 Increasing our Naïvety (Attempt 2)

Next, we double down on our first approach to illustrate how misguided it really is: instead of doubling the target memory space, we quadruple it. We take a

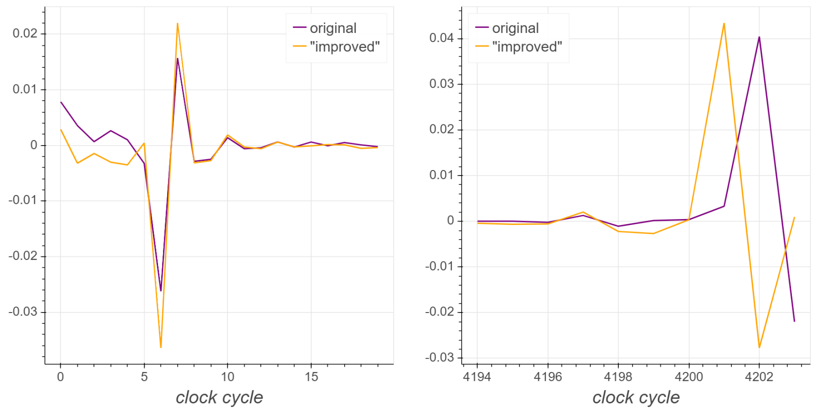

Fig. 9. Differences between the average power trace for $k_i = 1$ and $k_i = 0$.

second shot at hiding the leakage at cycles 4202–4203 by further uncoupling the memory write control logic from k. In the first attempt each half of the intermediate result memories played a static role (one always held good intermediate results, the other always held unused intermediate results). Now we quadruple the target memory space and alter the write destination logic so that the good intermediate results can go to any of the four memory sections, and ensure that the destination memory changes at every bit of k, regardless of its value.

The results are no better: the leakage markers become stronger still at cycles 4202–4203. With the DoM based attack, the average number of wrong bit guesses per trace goes down further to 15 bits per trace. Things are moving in the wrong direction! These unsuccessful results are included here to better reveal fundamental truths about the attacks and why they work. While it's a handy shorthand to refer to the leakage as originating from masked memory writes, it's crucial to understand that the leakage can be more complex than that.

4.3 Adding Noise (Attempt 3)

Instead of devising additional measures of increasing complexity to hide the leakage, we now take a completely different approach: adding "noise". The earlier improvements are abandoned—the k-dependent write logic and the target memories are returned to their original state. Instead, we add dummy logic which operates in tandem with the original leaky logic; the objective of this new logic is to add noise which hides the leakage.

We instantiate additional copies of the target memories. These copies are exercised with the same control logic as the real target memories, except that an LFSR is used to pseudo-randomly enable or disable the writes. The goal is for the noise memories to be active at the same time of the leakage, but in a way that does not depend on k. Experimentally, we find that adding a single "noise" memory for each of the 3 target memories does not help much, so we crank the noise up to 16 noise memories per target memory (48 in total).

The initial finding of leakage times is essentially unaffected by this countermeasure. When we look at the average between ones and zeros, we are comparing the average power trace segment of two groups of 128 bits from the same trace. This averaging over 128 bits allows the added noise to get "averaged out" quite well. However, when we move to single-trace attacks, each bit of k is treated individually and so no averaging is possible. Table 2 summarizes the results: with sufficient noise memories, this countermeasure is effective. There is some irregularity in the number of traces suitable for EHNP, stemming from the luck of where and when incorrect guesses occur (or lack thereof).

Number of active noise memories (per target memory)	Number of traces for full k recovery	Number of wrong bit guesses per single trace	Number of traces suitable for EHNP
1	5	26	6
8	6	39	18
16	15	68	2

Table 2. Attempt 3 results with 10000 traces.

4.4 Simple Balancing (Attempt 4)

In this final attempt we abandon attempts at small inexpensive changes: we go big by instantiating a full second copy of the target which operates synchronously with the original target. This second instance processes the bit-wise inverse of k. In theory, this should completely eliminate the leakage; in practice, due to variances in the physical placement of each instance's logic, the two implemented instances are not identical, and therefore their respective leakages cannot be expected to fully cancel each other out.

The leakage markers are still present where we expect to find them, and they remain fairly strong; however, Figure 10 shows that several stronger markers now appear throughout the full length of the power trace segment. These new markers are coincident with the much weaker markers observed with the unmodified target (Figure 4). Surprisingly, using these new markers as distinguisher components, we obtain an *extraordinarily strong* DoM-based attack which identifies the location of the most significant 1 and 0 of k. Figure 11 shows the result of using a *single trace* with the following k:

$k =$ `0x0000ffffffffff000000000000ffff00aaaa0000cccc0000111100003333 0000`

The initial steep drop is due to the leading 1 of the inverse of k that is processed by the duplicate core; the next steep drop is due to the leading 1 of k. Other bits of k are not revealed by this marker; only the leading 1 and leading 0 are revealed (that the leading 0 is also revealed highlights how easy it can be for countermeasures to have unintended effects!).

This leakage is not unique to this particular countermeasure attempt. Returning to the unmodified target implementation and using the newly found markers, we find this leakage to be just as strong. In Section 3 we ignored those markers because they are orders of magnitude weaker than the markers that were chosen. But perhaps because these weak markers are so numerous, they leak the leading 1 of k with a *very* high signal-to-noise ratio.

The source of this new leakage can be readily found with a quick look at a simulation waveform: we find that intermediate results written to the `bram_rz` target memory are `256'd1` (255 zeros followed by a single 1) as long as the target is processing leading zeros; when the first 1 is encountered, data written to `bram_rz` changes to random-looking data with a Hamming weight of around 128. This is not surprising: we can infer that the target converts the base point from affine to projective space by setting $z = 1$. This could be easily addressed, at very little cost, by randomizing the z coordinate (i.e. point blinding). While these results suggest that an HNP attack using the most significant bits could be *highly* effective, we don't pursue it because this leakage is easy to fix.

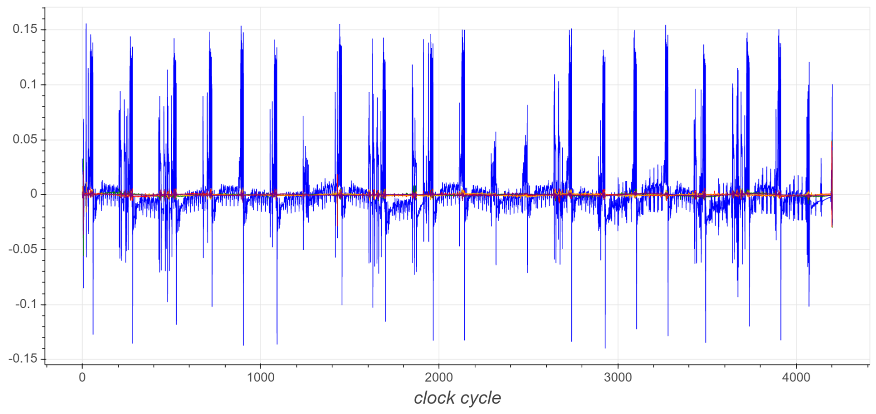

Fig. 10. Differences between the average power trace for $k_i = 1$ and the average power trace for $k_i = 0$, fourth attempt in blue, other attempts in different colours.

Returning to the DoM distinguisher based power attack used on the other attempts, we find on average 50 wrong bit guesses per single trace, and 4 traces out of 10000 have sufficient consecutive bit guesses for EHNP, which under performs attempt 3. Table 3 summarizes the results obtained thus far, using the DoM approach. Attempt 3 is the clear winner: it is both cheaper in resources and more effective than attempt 4. In terms of FPGA resource costs, there is no material difference between the original implementation and attempts 1 and 2. Attempt 3 consumes 1.3 times more lookup tables and 2.8 times more BRAMs, although these numbers are misleading because each noise memory is implemented as a 36 kilobit BRAM, of which only 256 bits are needed; this could

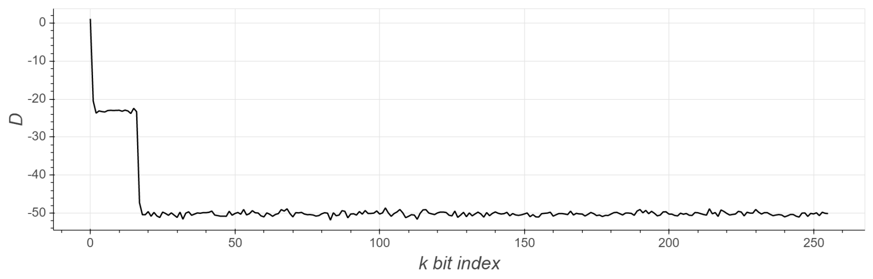

Fig. 11. A single trace reveals the location of the leading 1 and 0 of k.

be optimized without compromising the side-channel performance. Attempt 4 consumes 2 times more lookup tables and 1.8 times more BRAMs.

Implementation	Number of wrong bits guesses per single trace	Number of traces suitable for EHNP
Original	29	7
Attempt 1	21	15
Attempt 2	15	30
Attempt 3	68	2
Attempt 4	50	4

Table 3. Summary of all results with 10000 traces.

5 Test Vector Leakage Assessment

Test Vector Leakage Assessment (TVLA) is a method for evaluating the feasibility of side-channel attacks on implementations of cryptographic algorithms [12,30]. The use of TVLA has also been standardized in ISO/IEC 17825 [15]; it is a front-runner for usage as industry evaluation metric.

Since the purpose of TVLA is to detect secret-dependant points in the power traces, we are curious to see how well TVLA can identify the leakage that was used for our attacks. We carry out a TVLA test with P fixed and k variable. We use the t-statistic threshold of 4.5 to indicate a statistically significant difference between k being fixed or variable, and find that using a total of 10000 traces, failures occur; Figure 12 shows the TVLA results for the first three bits of k. The first failures occur around cycle 4210, which coincides with our DoM results. Beyond that, strong failures abound, giving a very different picture than the results of Figure 4. Results beyond the third bit of k resemble those of bit 3 and are omitted for clarity. The numerous TVLA failures are not surprising since k is directly used, unprotected, by the target, and power traces are perfectly aligned.

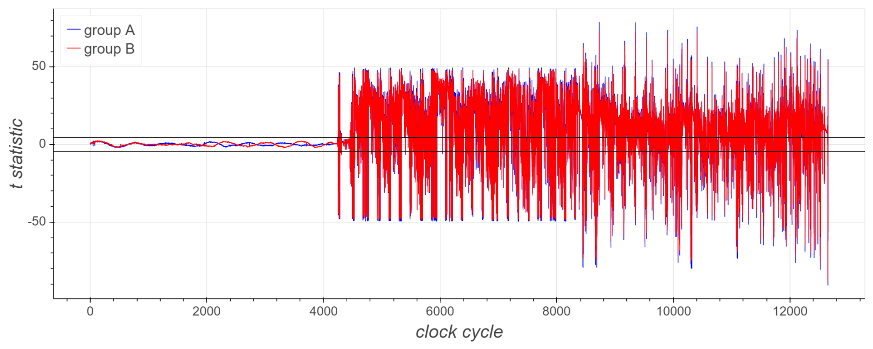

Fig. 12. TVLA results for the first three bits of k. The black horizontal lines indicate the failure threshold.

Figure 13 overlays the TVLA results for the second bit of k with the average difference between zeros and ones from attempt 4, using absolute values and scaling to better highlight their similarity. Whereas the peaks of the two curves coincide strongly, the TVLA peaks are much more indistinct. In particular, using t-test scores above 4.5 would not be useful at all for finding leakage in the source code, since almost every clock cycle would qualify.

The use of TVLA is primarily designed as a pass/fail metric for simple evaluation of cryptographic implementations to detect any data-dependant leakage. These results confirm that, as expected, TVLA does show data-dependant leakage, but other methods such as our DoM approach provide the fine-grained information on the actual source of the leakage that can be used to either build an attack, or mitigate the leakage.

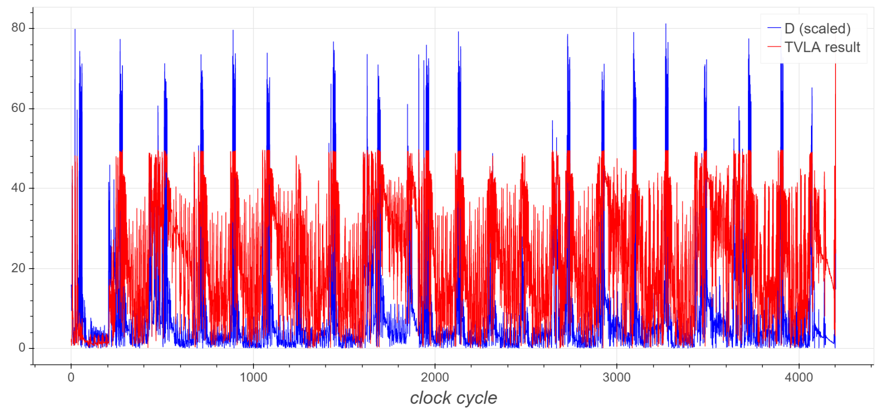

Fig. 13. TVLA results for k_2, and the difference between average power traces segments for zeros and ones of attempt 4.

6 Conclusions

Power analysis attacks on ECC are well known in both theory and practice [9,17,6,10,3]. Despite this long pedigree, the availability of open-source artifacts for hardware cores is limited, especially when compared to examples for AES. Such artifacts are useful for both academic and industrial research groups, and the security of hardware implementations of ECDSA has become more critical to evaluate due to the use of ECDSA in a variety of secure boot and RoT solutions.

In this work we used an existing open-source ECDSA implementation, and demonstrated not only how a power analysis attack can be built, but also how to connect the design information to the leakage to identify the source of the leakage. Compared to previous work, we have made extensive documentation and code available for recreating the entire attack chain in this paper. We explored several implementation-specific countermeasures and demonstrated how the measured leakage compares to the assumptions made in the countermeasure design. Work to automatically identify and correct side-channel leakage in hardware designs can also benefit from this artifact.

Several variants of this ECDSA core are available from CrypTech. Our analysis of the unprotected design variants shows similar leakage. Most recently CrypTech has included a Montgomery ladder implementation, which is commonly cited as a protection against SPA. Our initial evaluation of this version showed very similar leakage to what is shown in this paper, but the core is still in development (our results are being relayed back to CrypTech developers for their feedback).

Whether the side-channel results presented on our FPGA target maps to ASIC targets is another area to be explored—for example the use of BRAMs for small memories, which on an ASIC would likely be implemented in flip-flops. The use of an open-source ECC design makes it possible for future researchers and groups to compare their work to our results, whether they are implementing the designs in FPGAs or ASICs.

Acknowledgements

Thanks to the CrypTech project for providing the open-source ECDSA core that forms the basis for the artifact and makes this work possible. We are particularly in debt to Pavel Shatov who authored these cores, and with whom we discussed initial results of this work. The CrypTech core we evaluated did not claim side channel resistance and thus we must again emphasize this is *not* a flaw or threat in the CrypTech core or model. Instead we are extending the open-source nature of the CrypTech project to also serve as a hardware ECDSA implementation artifact. CrypTech is aware of these results and this work, and newer versions of the cores include side-channel countermeasures.

References

1. CrypTech 2015 End of Year Report. Tech. rep., CrypTech (Dec 2015), https://cryptech.is/wp-content/uploads/2015/12/CrypTech-EOYReport-2015.pdf
2. Side-Channel Marvels (2016), https://github.com/SideChannelMarvels
3. Abarzúa, R., Valencia, C., López, J.: Survey on performance and security problems of countermeasures for passive side-channel attacks on ECC. Springer Berlin Heidelberg, Berlin, Heidelberg (2021)
4. Abdulgadir, A., Diehl, W., Kaps, J.P.: An open-source platform for evaluation of hardware implementations of lightweight authenticated ciphers. In: 2019 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2019, Cancun, Mexico (Dec 2019)
5. Batina, L., Hogenboom, J., Mentens, N., Moelans, J., Vliegen, J.: Side-channel evaluation of FPGA implementations of binary Edwards curves. In: 2010 17th IEEE International Conference on Electronics, Circuits and Systems. pp. 1248–1251 (2010)
6. Belgarric, P., Fouque, P.A., Macario-Rat, G., Tibouchi, M.: Side-Channel Analysis of Weierstrass and Koblitz Curve ECDSA on Android Smartphones. In: Topics in Cryptology - CT-RSA 2016. pp. 236–252. Springer International Publishing (2016)
7. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: Hiding your white-box designs is not enough. In: Gierlichs, B., Poschmann, A.Y. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2016. pp. 215–236. Springer Berlin Heidelberg, Berlin, Heidelberg (2016)
8. Camurati, G., Poeplau, S., Muench, M., Hayes, T., Francillon, A.: Screaming Channels: When Electromagnetic Side Channels Meet Radio Transceivers. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 163–177. CCS '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3243734.3243802, http://doi.acm.org/10.1145/3243734.3243802
9. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems. pp. 292–302. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
10. Fan, J., Verbauwhede, I.: An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost, pp. 265–282. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
11. Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., Yarom, Y.: ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 1626–1638. CCS '16, Association for Computing Machinery, New York, NY, USA (2016)
12. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side channel resistance (2011)
13. Hlaváč, M., Rosa, T.: Extended hidden number problem and its cryptanalytic applications. In: Biham, E., Youssef, A.M. (eds.) Selected Areas in Cryptography. pp. 114–133. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
14. Hutter, M., Medwed, M., Hein, D., Wolkerstorfer, J.: Attacking ecdsa-enabled rfid devices. In: Abdalla, M., Pointcheval, D., Fouque, P.A., Vergnaud, D. (eds.) Applied Cryptography and Network Security. pp. 519–534. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
15. ISO/IEC: 17825. https://www.iso.org/standard/60612.html (2016)

16. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. pp. 388–397. CRYPTO '99, Springer-Verlag, Berlin, Heidelberg (1999), http://dl.acm.org/citation.cfm?id=646764.703989
17. Liardet, P.Y., Smart, N.P.: Preventing spa/dpa in ecc systems using the jacobi form. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems — CHES 2001. pp. 391–401. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
18. Lisovets, O., Knichel, D., Moos, T., Moradi, A.: Let's take it Offline: Boosting Brute-Force Attacks on iPhone's User authentication through SCA. Cryptology ePrint Archive, Report 2021/460 (2021), https://eprint.iacr.org/2021/460
19. Lomne, V., Roche, T.: A Side Journey to Titan. Tech. rep. (Jan 2021), https://ninjalab.io/wp-content/uploads/2021/01/a_side_journey_to_titan.pdf
20. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Advances in information security, Springer (2008)
21. Mangard, S., Pramstaller, N., Oswald, E.: Successfully Attacking Masked AES Hardware Implementations. In: Rao, J.R., Sunar, B. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2005. pp. 157–171. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
22. Moradi, A., Schneider, T.: Improved side-channel analysis attacks on xilinx bitstream encryption of 5, 6, and 7 series. Cryptology ePrint Archive, Report 2016/249 (2016), https://eprint.iacr.org/2016/249
23. O'Flynn, C., Chen, Z.: Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection. Journal of Cryptographic Engineering **5**, 53–69 (2013). https://doi.org/10.1007/s13389-014-0087-5
24. O'Flynn, C., Chen, Z.: ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. IACR Cryptology ePrint Archive p. 204 (2014)
25. Oswald, D., Richter, B., Paar, C.: Side-Channel Attacks on the Yubikey 2 One-Time Password Generator. In: Proceedings of 16th Symposium on Research in Attacks, Intrusions, and Defenses (RAID '13). Lecture Notes in Computer Science, vol. 8145, pp. 204–222. Springer Berlin Heidelberg (2013)
26. Oswald, E., et al.: OpenSCA, an open source toolbox for Matlab (2010), https://sourceforge.net/projects/opensca
27. Ravi, P., Jungk, B., Jap, D., Najm, Z., Bhasin, S.: Feature selection methods for non-profiled side-channel attacks on ecc. In: 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP). pp. 1–5 (2018). https://doi.org/10.1109/ICDSP.2018.8631824
28. Ronen, E., Shamir, A., Weingarten, A., O'Flynn, C.: IoT Goes Nuclear: Creating a ZigBee Chain Reaction. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 195–212 (2017)
29. San Pedro, M., Servant, V., Guillemet, C.: LASCAR: Ledger's advanced side channel analysis repository (Dec 2018), https://github.com/LedgerHQ/lascar
30. Tunstall, M., Goodwill, G.: Applying tvla to public key cryptographic algorithms. Cryptology ePrint Archive, Report 2016/513 (2016), https://eprint.iacr.org/2016/513
31. Varchola, M., Drutarovsky, M., Repka, M., Zajac, P.: Side channel attack on multiprecision multiplier used in protected ecdsa implementation. In: 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig). pp. 1–6 (2015)

32. Vasselle, A., Maurine, P., Cozzi, M.: Breaking Mobile Firmware Encryption through Near-Field Side-Channel Analysis. In: Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop. p. 23–32. ASHES'19, Association for Computing Machinery, New York, NY, USA (2019)