IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 2, pp. 166–191. DOI:10.46586/tches.v2022.i2.166-191

Side Channel Attack On Stream Ciphers: A Three-Step Approach To State/Key Recovery

Satyam Kumar[1], Vishnu Asutosh Dasu[2], Anubhab Baksi[3], Santanu Sarkar[1], Dirmanto Jap[3], Jakub Breier[4,5] and Shivam Bhasin[3]

[1] Department of Mathematics, Indian Institute of Technology Madras, Chennai, India
[2] TCS Research and Innovation, Bangalore, India
[3] Nanyang Technological University, Singapore
[4] Silicon Austria Labs, TU-Graz SAL DES Lab, Graz, Austria
[5] Graz University of Technology, Graz, Austria

ma17d002@smail.iitm.ac.in, vishnu.dasu1@tcs.com, anubhab001@e.ntu.edu.sg, santanu@iitm.ac.in, djap@ntu.edu.sg, jbreier@jbreier.com, sbhasin@ntu.edu.sg

Abstract. Side Channel Attack (SCA) exploits the physical information leakage (such as electromagnetic emanation) from a device that performs some cryptographic operation and poses a serious threat in the present IoT era. In the last couple of decades, there have been a large body of research works dedicated to streamlining/improving the attacks or suggesting novel countermeasures to thwart those attacks. However, a closer inspection reveals that a vast majority of published works in the context of symmetric key cryptography is dedicated to block ciphers (or similar designs). This leaves the problem for the stream ciphers wide open. There are few works here and there, but a generic and systematic framework appears to be missing from the literature. Motivating by this observation, we explore the problem of SCA on stream ciphers with extensive details. Loosely speaking, our work picks up from the recent TCHES'21 paper by Sim, Bhasin and Jap. We present a framework by extending the efficiency of their analysis, bringing it into more practical terms.
In a nutshell, we develop an automated framework that works as a generic tool to perform SCA on any stream cipher or a similar structure. It combines multiple automated tools (such as, machine learning, mixed integer linear programming, satisfiability modulo theory) under one umbrella, and acts as an end-to-end solution (taking side channel traces and returning the secret key). Our framework efficiently handles noisy data and works even after the cipher reaches its pseudo-random state. We demonstrate its efficacy by taking electromagnetic traces from a 32-bit software platform and performing SCA on a high-profile stream cipher, TRIVIUM, which is also an ISO standard. We show pragmatic key recovery on TRIVIUM during its initialization and also after the cipher reaches its pseudo-random state (i.e., producing key-stream).

Keywords: Stream Cipher · Side Channel Attack · Machine Learning · Mixed Integer Linear Programming · Satisfiability Modulo Theory · Feedback Shift Register

1 Introduction

Symmetric key cryptography is among the cornerstones in ensuring security in present-day electronic communication. The symmetric key ciphers are typically highly efficient

We wish to thank François-Xavier Standaert and the anonymous reviewers for their detailed comments and helpful suggestions.

Received: 2021-10-15 Accepted: 2022-12-15 Published: 2022-02-15

compared to their asymmetric counterparts. Consequently, symmetric key ciphers are used whenever the communication protocol allows it.

Therefore, understanding the potential threats that challenge the security of the symmetric key ciphers is of prime importance. The security of a such a system is usually challenged on two aspects. The first aspect, termed as *classical attack*, rigorously analyses the algorithmic structure. In contrast, the second aspect relies only on the physical characteristics of a device which is running the cipher. Thus, it bypasses the ingenious construction that governs the security against the classical attacks. In this work, we concentrate on one such class of a device dependent attack, known as the *Side Channel Attack* (SCA, for short) [KJJ99, Koc96, MOP07, Pee13]. In this case, the *attacker*, observes physical characteristics such as timing, power consumption, electromagnetic emanation and so on. When the secret component of the cipher (typically termed as the *key*), takes part in the process, it influences the external characteristics of the device. Equipped with this knowledge, the attacker is commonly able to deduce some non-trivial information regarding the key. There is another type of a device dependent attack, the so-called *Fault Attack* (FA) [BS97].

Being a major concern for the security, SCAs have garnered serious attention from the cryptographic community. Still, it appears that the block ciphers (and similar constructions) get the limelight practically all the time. Effectively, the other main branch of the symmetric key cryptography that deals with stream ciphers (and similar constructions) is apparently under-analysed (see Section 2 for a quick review of the existing literature). In this work, we look for potential solutions to the problem of finding a suitable model for stream ciphers (and similar constructions), using which an efficient side channel analysis would be possible. The framework we present here offers multiple features, e.g., it is an automated framework which can deal with real (noisy) traces for the Hamming weight (software) and Hamming distance (hardware) leakage model. Additionally, it works even after the cipher reaches its pseudo-random phase (we also use term key-stream phase interchangeably).

To show the usefulness and non-triviality of our work, we point to the following quotes:

- ◇ "Knowledge about the Hamming weight of intermediate values do probably give enough information to exclude some possible keys and therefore to reduce the key space. However, Hamming weight information by itself is often not sufficient to derive the secret key." – [RO04, Section 1]
- ◇ "As there is no obvious way to recover the internal state of the cipher after the initialization phase when the key is spread amongst the 288 bits of internal state [of `TRIVIUM`], applying any kind of side channel attack at a later stage is not promising." – [GBC+08, Section 4.6]

Our Contribution

The challenge we partake here is to design a generic framework that can retrieve the key of a stream cipher or a similar construction given the side channel leakage, with minimal human intervention. Moreover, the framework should be able to retrieve non-trivial information from the leakage during the pseudo-random generation algorithm, and account for noise present in the leakage. The source-code is available as an open-source at the bitbucket repository[1]. While more information about our modelling is given in Section 3, the work-flow can be summarised as follows (see Figure 1 for a pictorial description):

1. **Offline Stage:**

   (a) Get the side channel traces from the target device. Using the proper leakage model, for example Hamming weight for software or Hamming distance from hardware, those traces can be formulated as a multi-class classification problem. For instance, the traces can be formulated as a classification problem with

[1] https://bitbucket.org/anubhab001/trivium-3step-sca/.

33 classes when the leakage is from a 32-bit microcontroller since 33 distinct Hamming weights are possible.

(b) Now the problem can be trained with an appropriate *Machine Learning* (ML) model that supports classification.

(c) Test SMT solver for its tolerance limit, t_l by using simulated noisy information.

2. **Online Stage:**
   (a) With the trained ML model, we estimate the class in which the target traces would belong. Thus, we can get some non-trivial information in the form of Hamming weight or Hamming distance from the traces.
   (b) This enables us to fit a *Satisfiability Modulo Theory* (SMT). If all the predictions from the ML are correct, then the SMT instance returns a solution for the unknown state/key in a reasonable time.
   (c) Since the probability for correct prediction of HW through the ML model is not 1, we change the definition of correct prediction. Let c be the original class, then for a given $\epsilon > 0$, a prediction c' of the class is correct if $c' - \epsilon \leq c \leq c' + \epsilon$ holds, otherwise the prediction is wrong.
   (d) Since the prediction from the ML model (Step 2(a)) is not 100% accurate for any given $\epsilon < 7$, once in a while a wrong prediction will make it way through the SMT instance. This is problematic, since just 1 wrong prediction makes the entire system inconsistent (even though all the rest predictions are correct). To filter that, we use another model, which is based on *Mixed Integer Linear Programming* (MILP). This MILP instance takes the sequence of predictions from the ML model, and uses cipher specific information, such as, the Hamming weight of one block at a certain clock cannot be too dissimilar to that of the same at the next clock. With a high probability, this MILP instance returns a sequence with no anomaly.
   (e) The output from MILP is finally fed into the SMT instance (Step 2(b)), which is then solved with error tolerance, t_l (Obtained in Step 1(c)).

In actuality, the modellings (ML, MILP and SMT) are more complicated, this comes from the innate nature of the side channel traces. With the naïve ML modelling, we observe that the class-prediction accuracy is quite low. This leads to discard of too many traces (as SMT needs all correct traces), consequently the same experiment is to be repeated an impractical number of times. To mitigate the problem, we introduce the concept of *error tolerance* (denoted by ϵ) which allows for a middle ground. More information regarding the concept of tolerance can be found in Section 3.2.

A practical evaluation of our framework is described with the TRIVIUM [DCP08] (description of TRIVIUM is omitted here due to space constraint) stream cipher running on an ARM Cortex-M3 board (32-bit microcontroller) in Section 4 (some additional results on 8- and 16-bit microcontrollers are also shown). In particular, Section 4.2 describes the set-up used, Section 4.3.1 discusses about the machine learning related results, the MILP model that corrects the output sequence given by ML is discussed next in Section 4.3.2, following which the SMT model is described in Section 4.3.3. In Section 4.3.4, we summarise the SMT module. Thereafter, our experimental results are given in Section 5. Results on the Hamming weight model (software) are given in Section 5.1 (Section 5.1.1 for pseudo-random phase and Section 5.1.2 for initialisation phase), and on the Hamming distance (hardware) model are given in Section 5.2. To test our framework for different SNR, we put results on success probability of our framework with respect to different SNR using a simulated traces in Section 6. Lastly, we conclude in Section 7.

2 Related Works

Coming to the side channel attacks on stream ciphers and related constructions, Rechberger et al. [RO04] present a detailed survey of the attack on stream ciphers and countermeasures in 2004. Fisher et al. [FGKV07] in 2007 propose a DPA on GRAIN-v1 [HJM07] and TRIVIUM [CP08]. Their attack is carried out using multiple chosen IVs. Both attacks are done in the initialisation phase. To remove the noise, they suggest to average out power traces with identical input parameters. Strobel et al. [Str09] carry out a side channel attack based on algebraic method on GRAIN and TRIVIUM using multiple IVs in the initialisation phase. In 2008, a report on susceptibility of eSTREAM candidates is submitted by Gierlichs et al. [GBC$^+$08]. They show the vulnerabilities of the eSTREAM ciphers against various SCAs. They suggest an attack on TRIVIUM and GRAIN-v1 during the initialisation only. Later some SCA on stream cipher is carried out, such as; [HYY$^+$10] on K2; [QGGL13] on CRYPTO-I (an LFSR-based ciphers); and [KAA$^+$17] on CRYPTO-I, TRIVIUM, GRAIN and BIVIUM-B. All of these assume noiseless traces and needed multiple IVs for their attack. In 2015, Chakraborty et al. [CMM15] propose a side channel attack on GRAIN family of stream ciphers using multiple IVs in the initialisation phase. Tena-Sánchez et al. presents two papers [TA15, TSA15] in 2015 on TRIVIUM. Both the papers target the initialisation phase and need at least 1200 different IVs but can deal with noisy traces.

In a recent work by Sim, Bhasin and Jap in TCHES'21 [SJB20] (which we refer to as DAPA for simplicity), the authors show that for TRIVIUM a partial key recovery (66 out of 80 bits) is possible from the initialisation phase, with the assumption that the obtained side channel leakage is noise-free.

Indeed, we observe that most of the relevant research works only target the initialisation phase. To the best of our knowledge, the only existing work to address the attack on the pseudo-random generation phase is [KAA$^+$17]. This attack, however, is specific to the CRYPTO-I cipher; and probably does not scale up for mainstream ciphers like TRIVIUM[2]. The difficulty of recovering state bits in the pseudo-random phase is that one can no longer use IV information as it is mixed up completely in the psuedo-random phase. A comparative summary of the literature survey with our work is shown in Table 1.

Table 1: A comparative study of our work with other stream cipher SCAs

Research Work	Year	Target Cipher(s)	Attack Phase	Noisy?	# IV?
Fischer et al. [FGKV07]	2007	GRAIN, TRIVIUM	Initialisation	Noisy	Multiple
Gierlichs et al. [GBC$^+$08]	2008	eSTREAM Candidates	Initialisation	Noisy	–
Strobel [Str09]	2009	GRAIN-v1, TRIVIUM	Initialisation	Noisy	Multiple
Qu et al. [QGGL13]	2013	CRYPTO-I	Initialisation	Noiseless	Multiple
Chakraborty et al. [CMM15]	2015	GRAIN Family	Initialisation	Noisy	Multiple
Tena-Sánchez et al. [TA15, TSA15]	2015	TRIVIUM	Initialisation	Noisy	Multiple
Kazmi et al. [KAA$^+$17]	2017	CRYPTO-I	Pseudo-random	Noiseless	Single
		TRIVIUM, BIVIUM-B, GRAIN	Initialisation	Noiseless	Single
Jurecek et al. [JBL19]	2019	A5/1	Initialisation	Noiseless	–
Sim et al. [SJB20]	2020	LR-KEYMILL, TRIVIUM	Initialisation	Noiseless	Multiple
Our Paper	2022	TRIVIUM	Initialisation, Pseudo-random	Noisy	Single

In some sense, our work can be associated with *Algebraic Side Channel Attack* (ASCA). Some major results related to ASCA on block ciphers can be found in [Jaf07, RS09, CFGR12, RSV09, ORSW12]. Their method to model HW as a SAT equation (described in [CFGR12]) will have a large complexity in the case of stream ciphers with more than billions of clauses because of a larger state size. ASCA uses heuristic approach, solving the system of equations using the side channel information. However, the method assumes that the side channel information is decoded and presented perfectly. In a real setting, the side channel information is noise prone, and the side channel decoding might not

[2]In the same work, the authors show a side channel attack on TRIVIUM during the initialisation phase.

be error-resilient. Several improvements for ASCA have been proposed, to mitigate the error issue, such as extending the equation to accept set of possible solutions, as shown in *Set-ASCA* [RSV09, ZWG+11] and *Improved ASCA* (IASCA) [MBZ+13], or by using optimizer to solve imprecise deduction in the equation, as demonstrated by *Tolerant ASCA* (TASCA) [OKPW10, ORSW12]. The latest approach is to exploit the soft information provided by profiled side channel attacks, commonly referred as the *Soft-Analysis SCA* (SASCA) [VGS14, GGSB20]. Though powerful, SASCA [BCS21] can be quite demanding in terms of profiling required. However, as per our knowledge, all these attacks have only been demonstrated for block ciphers.

3 Attack Methodology

3.1 Brief Overview of the Framework

As noted already in Section 1, our paper presents an automated framework which is able to recover the unknown state or key bits of a stream cipher (or related constructions) given the side channel information from a device. Our framework is generic, as it can target any type of device (software or hardware), can work with a variety of ciphers that follow the basic *Non-linear Feedback Shift Register* (NLFSR) based construction (most notably, stream ciphers), considers the real leakage from a device (i.e., does not use simulated traces which are noiseless), needs only one execution of the cipher (thereby respecting the key – IV uniqueness), and can attack initialisation and pseudo-random phases alike. It involves three solvers (ML, MILP and SMT) that work in conjunction with the rest and returns the output in a reasonable time.

From a top level view, the overall framework can be conceptually spilt into two steps:

(1) *Predict Hamming weight/distance from traces.* We need to get a good estimate of the Hamming weight (HW) on software or Hamming distance (HD) on hardware, given the leakage information. For simplicity and conciseness, here we focused on HW model initially. Later on we took a step forward to show that our SMT model works for Hamming distance model also.

(2) *Fit an automated tool to retrieve key.* Next, we need to fit the side channel information obtained to an automated tool which will eventually compute the secret key. If we are targeting the pseudo-random generation stage of the cipher, the key-stream is available, and that can be used in the solution process too.

We intuitively choose ML (as it has native support for classification) for Step (1) and SMT for Step (2). However, the overall method turns out to be considerably more complex than just that, as described in Section 3.2.

3.2 Description of the Framework

This basic overview of our framework given in Section 3.1 suffers from two major problems. The relevant discussions, along with the respective remedies, are given next.

3.2.1 Low Accuracy by Machine Learning Model

The first major problem comes from the difficulty of classification of Hamming weight/distance from the ML instance. This happens due to the relative proximity of one class to the others. With the naïve implementation, the ML accuracy turns out to be little less than 40% (for more details of our ML model, see Section 4.3.1), despite dedicated efforts to increase it.

Having such a low accuracy is a problem, as the later part that takes these predictions and tries to solve for the unknown components, needs to be fed with 100% accurate predictions. Only one wrong prediction would potentially make the entire SMT system inconsistent. An initial estimate suggests that we would need a few hundred predictions for HW/HD for a typical cipher like TRIVIUM [DCP08]. This means, one has to run an impractical number of independent experiments in order to get all correct predictions from the ML model.

To find a way to increase the accuracy of the ML model, we make use of the following interesting observation. We notice that, given the correct class is c, the model will more likely return a class from $\{c-\epsilon, \dots, c, \dots, c+\epsilon\}$ than c for $\epsilon \geq 1$. In other words, it is highly likely that the ML model will predict a class within the proximity of the actual class (than it will predict the actual class). This gives us the idea of tolerance, ϵ, which can be interpreted as follows (consider the correct class is c):

(i) When $\epsilon = 0$, the accuracy of the ML prediction is counted as-is (i.e., we count the case as accurate if predicted class is c).

(ii) When $\epsilon > 0$, we count the case where the predicted class belongs from $\{c - \epsilon, \dots, c, \dots, c+\epsilon\}$ as accurate.

Intuitively, with the increase of ϵ, the corresponding (custom definition of) accuracy will increase. At the same time, the SMT instances can be tweaked to work with a more flexible predictions (that contain the tolerance ϵ). More specifically, any SMT constraint of the form, $x = c'$ can be reformulated with ϵ as, $c' - \epsilon \leq x \leq c' + \epsilon$ (x can be, for example, the HW of the state and c' is the predicted class). Thus, the concept of error tolerance, ϵ, allows us to increase accuracy for the ML prediction, and is compatible with SMT modelling.

Table 2: Trade-off between ML tolerance and SMT solution time (sec.) for TRIVIUM

	Tolerance (ϵ)				
	0	1	2	3	4
ML Accuracy	0.3933	0.86678	0.98262	0.99784	0.99967
SMT Solving Time	5.42 (110)	254.36 (110)	1819.21 (130)	28755.36 (170)	76797.41 (130)

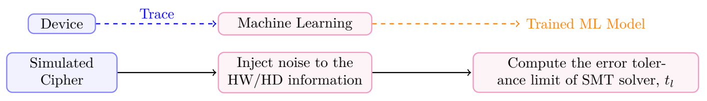
$(\cdot)$: Number of TRIVIUM pseudo-random rounds considered

However, a trade-off for tolerance is to be made. Making ϵ larger will increase the ML prediction accuracy, but will make the SMT instance less effective. Ultimately, beyond a certain threshold for a given platform, the SMT solver will take impractical time to solve, rendering it useless. In our experiment with the ARM Cortex M3, we observe that the ML model achieves accuracy of 100% starting tolerance $\epsilon = 7$ (Table 3). For this level of tolerance, the formulated SMT instances appear to take too much time for our target cipher TRIVIUM (more results can be found in Section 4). This implies, we need to reduce ϵ to get a solution from the SMT problem in a reasonable time, at the expense of reduction of accuracy for the ML model (unless high performance computing resources are used). Reducing ϵ further to 4, we observe that the corresponding SMT instance is solvable in our set-up, but the solution time is around 13 to 21 hours. Finally, we settle down at $\epsilon = 3$, where the ML accuracy is 99.784%, which is quite high, and also the SMT instances can be solved on an average of 10 hours. This choice of ϵ appears to hit the sweet-spot for the conflicting requirements. An instance for the scenario can be seen from Table 2, where the number in the parenthesis represents the number of rounds[3] used for SMT modelling with TRIVIUM during its pseudo-random phase (key-stream information is used in the solution process).

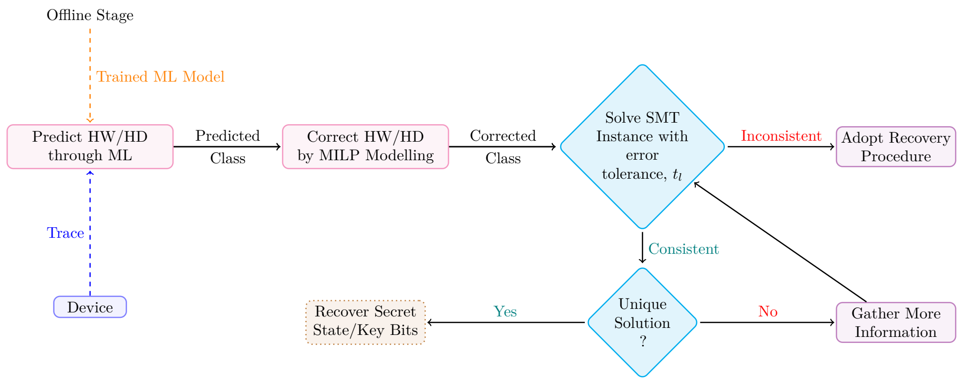
3.2.2 No Resilience by SMT Model

If we could use a higher tolerance (e.g., $\epsilon \geq 7$ in Section 3.2.1) so that the ML prediction works with 100% accuracy, then we would be certain that all the information passed to

[3]With our repeated experiments, the number of rounds shown here appear optimal in terms of solution time for the respective category.


Device
Trace
Machine Learning
Trained ML Model
Simulated Cipher
Inject noise to the HW/HD information
Compute the error tolerance limit of SMT solver, t_l

(a) Offline



(b) Online

Figure 1: Schematic of work-flow of our side channel attack framework

the SMT problem is error free. However, as we have seen, increasing the tolerance too high would also mean the SMT problem is getting less information, that leads to more solution time for the solver.

Although the choice for ϵ (which is 3 in our case) allows for a very high accuracy for the ML model (99.784%), it is not 100%. This means, about 3 in 1000 predictions on average will be wrong (wrong in this context means, those predictions are outside the desired tolerance). As mentioned in Section 3.2.1 already, this is a problem since one wrong prediction would render the whole SMT instance inconsistent. There is no way for the attacker to know if/when a system is inconsistent until the SMT instance returns inconsistent, and even after that the attacker does not know which prediction(s) is wrong. This, in a way, leads us back to the same problem of Section 3.2.1. However, now the difference is, with the introduction of (non-zero) tolerance, the prediction accuracy for ML is quite high. This suggests that there could be redundant information so that an error correction would be feasible.

3.2.3 MILP Model for Correction of Wrong Predictions

As mentioned earlier in Section 3.2.1, a generic MILP model attempts to correct the ML predicted sequence so that all predictions are within the desired tolerance limit (so that the final model based on SMT does not return inconsistent[4]).

Indeed, keeping in mind that the ML model does not take into account the cipher-specific information (predicts the classes independently), an additional modelling can be put in place. Here we make use of the following types of information for a b-bit software platform (b consecutive bits of the state locations are updated at one clock, which are indicated as *block*):

(A) *Change of HW for entire state from a clock to next.* Observe that maximum increase in HW of the internal state for any consecutive rounds is equal to the number of incoming bits to the internal state during state update. Similarly, maximum decrease in the HW of a internal state is equal to the number of outgoing bits. Also, since

[4] Inconsistency means the constraints that are fed to the SMT instance are mutually contradictory.

the microcontroller splits the state and runs the blocks consecutively, change in Hamming weight of each such block from round t to $t+1$ depends upon the number of incoming and outgoing bits of that block.

(B) *Interaction among the blocks.*
- If we encounter any block of the type shown in Figure 6(a), then after b rounds (size of microcontroller used), HW of one block will be transferred to the next/previous block.
- One bit from previous block is shifted to the next block on the right side. This forms a conditional constraint, i.e., change in HW of next block depends upon whether the incoming bit is 1 or 0.

To capture the observations in (A) and (B), we devise a third model. It takes the sequence of predictions by the ML model, and formulates an MILP problem (described in Section 4.3.2) which returns (possibly) a slightly modified sequence so that all the predictions are within the desired tolerance. Finally, the output from this MILP instance is given to the SMT solver, which solves for the unknown state/key of the cipher (see Section 3.2.4 for a discussion on this).

This MILP model works as follows. It takes a sequence of predicted classes from ML as an input and returns a sequence of classes, which is not far apart from the predicted classes, and which respects the internal cipher structure (noted in (A) and (B)). The catch, however, is that it may not work all the time, i.e., the returned sequence may fall outside the desired tolerance limit. Empirically, we observe out of 1000 independent trials that (here $b = 32$), the returned sequence of classes lies within the desired tolerance limit in 976 trials[5] (see Section 4.3.2 for more details). That means, though the MILP succeeds most of the time, there is a probability of about 0.024 that the constraints to the SMT problem will be inconsistent.

3.2.4 Overall Construction of Our Framework

With all the information presented thus far, now we are able to describe a complete work-flow for our framework, a visualisation of it is given in Figure 1. During the offline (pre-processing) phase, we prepare the training of our ML model with the side channel traces. After this, during the online stage (when the actual attack is underway), first this ML model is queried with the observed sequence of traces and predictions from this model are noted with a certain predefined tolerance, ϵ. This predictions are done independently for each traces, thus it fail to account for the internal structure of the cipher. By capturing those specifics, we are able to ensure the sequence of predictions fall within the predefined ϵ with an MILP modelling (with high probability). This sequence is finally fed to an SMT solver that gives the solution for the state/key of the cipher.

Recovery Procedure for Inconsistent SMT Instance In the unlikely case that the SMT instance is inconsistent, we recommend repeating the attack in a combination of the following ways (recommendation in (β) is probably the most practical):

(α) Retrain the ML model and run the prediction with same traces as input. As the initial parameters of the ML model are randomised, it may return a different sequence which could resolve the issue.

(β) While collecting the traces, collect them for additional number of rounds. As shown in Section 4, we realistically need little over 100 rounds of traces. If, say, the trace is collected for about 500 rounds, some other part can be potentially targeted if one part seems not fixable by the MILP model. Since the success probability of MILP is quite high, we will most likely get success within 1 or 2 trials.

(γ) Collect the traces from the device again[6].

[5]In the process, the MILP model alters the ML predicted sequence at 62% places, on an average.

[6]If needed, IV can be altered to respect the uniqueness of (key, IV).

However, in MILP, increasing the number of classes decreases the success probability and for SMT the solution time increases after a threshold. Therefore, a lower number of classes is beneficial for our framework, provided SMT instances are solvable. A similar idea was also explored in the context of block ciphers in [RSV$^+$11].

Note on State/Key Recovery In our attack on pseudo-random generation phase (where the cipher is ready to produce key-stream or tag), we are able to recover the (unknown) state. Depending on the internal structure, (full) key recovery may or may not be possible. For our exemplary case with TRIVIUM, the state is invertible, thus it can be reverted back to the key/IV loading phase, finally enabling us to recover the key. In certain cases, however, the state update is not invertible, in that case (full) key recovery may not be possible, for example LIZARD [HKM17].

4 Evaluation of Our Framework

4.1 Configuration

1. **Cipher:** The high-profile stream cipher, TRIVIUM [DCP08], which is chosen as a final round candidate in the eSTREAM project[7], and also is standardized in ISO/IEC 29192-3[8]. Consequently, this cipher has received a large number of analysis, see [HLM$^+$20, SMB17] for examples. It is chosen by the authors of DAPA [SJB20] as well. In DAPA, the authors recover 66 (out of 80) bits of the secret key, the rest 14 bits are to be found by exhaustive search. As a direct comparison, we are able to retrieve the full (80 bits) key of TRIVIUM.

2. **Target Device:** We take electromagnetic traces from ARM Cortex-M3, a 32-bit microcontroller (refer to Section 4.2 for more information on the setup). The corresponding attack for a 16-bit or 8-bit microcontroller are easier, as we show in Sections 5.1.1 (for key-stream phase) and 5.1.2 (for initialisation phase, this is typically easier to solve). We also show some results related to hardware leakage in Section 5.2.

3. **Tools:** For ML (Section 4.3.1), we use a *Multi-layer Perceptron* (MLP) due to its simplicity, with PyTorch[9]. Gurobi[10] solver is used to solve MILP instances (Section 4.3.2). Finally, as for the SMT solver (Section 4.3.3), we choose Z3[11].

4.2 Experimental Setup

For the experiments, we implemented the targeted operation in assembly on Arduino DUE (ARM Cortex-M3) with 512KB flash, 96KB SRAM and 84 MHz operating frequency. We use Riscure high precision EM probe to capture the leakage, on a Lecroy WaveRunner 610zi oscilloscope. A preliminary test with known fixed data is conducted to identify the best measurement spot on the board. We first measure the leakage by running grid search across the target device, and calculate the SNR [MOP07, BDGN13] for each position. Then, we choose the spot with the best SNR leakage shown in Figure 2 (i.e. SNR $\approx$ 3.3). For profiling, we use stratified sampling to ensure all the HW classes are represented in the experiment. Around 33000 traces are collected with each stratum of HW of 0 to 32 (each HW corresponds to 1000 traces). Overall 2362000 $\approx 2^{21.17}$ traces are used.

[7]https://www.ecrypt.eu.org/stream/project.html.
[8]https://www.iso.org/standard/56426.html.
[9]https://pytorch.org/.
[10]https://www.gurobi.com/.
[11]https://pypi.org/project/z3-solver/.

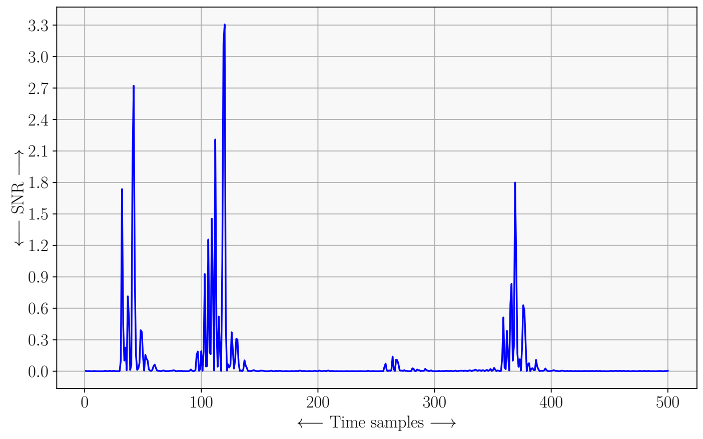


Figure 2: SNR for the measurement setup

4.3 Hamming Weight (Software) Model

4.3.1 Part I: ML to Predict Classes From Traces

With the experimental set-up described in Section 4.2, here we describe the results related to machine learning. This is a classification problem with 33 classes (as the number of HW for the target microcontroller will lie in $[0, 32]$) based profiled SCA (supervised learning, in context of ML). This kind of usage of ML in SCA is not uncommon, one may refer to [PHJ+17] for example. We choose MLP for the classification problem owing to its simplicity, though other ML models could be used. We collect 1000 traces for each of the HW classes for profiling. So in total 33000 traces are collected. After that, we collect more samples for the ML training. The total data-set size (training + validation + testing) for the ML training, including the traces for profiling, is $2362000 \approx 2^{21.17}$. We note that all classes are represented in the $2^{21.17}$ samples; but the data-set is imbalanced, i.e., the class distribution is unequal.

MLP models with input and output layer of size 500 and 33 respectively are trained with a 62.5/12.5/25 training/validation/testing split and a batch size of 64. All the experiments are done with Python 3.8.5, PyTorch 1.8.1+cu102 and Numpy 1.20.3 on a server with an Intel Xeon Platinum 8260 CPU, NVIDIA Quadro GV100 32GB GPU, 256GB RAM and Ubuntu 20.04.1 LTS as the operating system.

ReLU is used as the activation function after the hidden layers. AdamW [LH17] with learning rate 0.0001 is used as the optimizer with weighted cross entropy loss as the loss function to account for class imbalance. During training if overfitting is detected, the training process is terminated prematurely and the models are discarded. The results from fully trained models are indicated in Table 3.

We experiment with other activation functions, as well as deeper networks. However, we observe that the performance in all the other models is comparable to, if not worse, than the 2 hidden layer model. Moreover, the 2 hidden layer model works efficiently due to its low depth.

Optuna We also experiment with the automatic hyper-parameter optimisation framework, Optuna[12], to determine the optimal number of hidden layers and hidden layer size (following [KEI+21, Section 4]). A data-set of size $2^{19.345}$ with a 80/20 training/validation split is

[12]https://optuna.org/.

used. The number of trials is set to 500 and default settings are kept. At the end of the optimisation process, Optuna generates a model with 3 hidden layers each of size 416 as the optimal model with 569121 parameters. This model when retrained on the full data-set produces results similar to our 2 layer model with 84897 parameters as indicated in Table 3. During the optimisation process, we observe that Optuna would pick the same set of hyper-parameters multiple times; each time with slightly different validation accuracy, without pruning the trial. The reported accuracy in the Table 3, for the different data splits, is the average accuracy of all batches in an epoch rounded to 5 decimal places.

Table 3: Results for machine learning prediction (with varying tolerance)

	Train.	Val.	Testing Accuracy with Tolerance(ϵ)								Train.
	Acc.	Acc.	0	1	2	3	4	5	6	≥ 7	Time (s)
MLP-I	0.3601	0.39389	0.3933	0.86678	0.98262	0.99784	0.99967	0.99991	0.99998	1.0	5530.93
MLP-II	0.3661	0.3931	0.39266	0.86615	0.98234	0.99784	0.99965	0.99992	0.99999	1.0	6340.84

MLP-I: (MLP/ReLU) Hidden Layers (128, 128)

MLP-II: (MLP/ReLU) Hidden Layers (416, 416, 416), obtained from Optuna

Data size (training, validation, testing) = ($2^{20.494}, 2^{18.172}, 2^{19.172}$), Epochs = 100

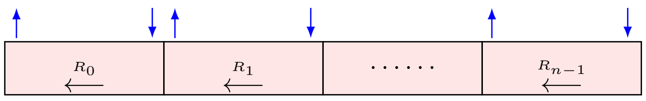
All in all, we conclude that our model with 2 hidden layers (with 128 neurons per layer) performs optimally compared to the models returned by Optuna. Also, with this MLP, the accuracy (when tolerance, $\epsilon = 0$) that we observe is just shy of 40%.

4.3.2 Part II: Correction of ML Predictions by MILP

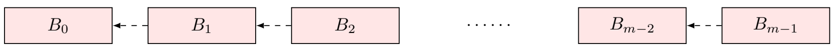
As discussed in Section 3.2.3, we need a model that can correct the sequence of ML predicted classes. MILP appears to be more efficient than SMT for this purpose.

Observe from Table 3, that increasing tolerance improves accuracy but on the other hand it also increases SMT solution time (check Table 2). In `TRIVIUM`, the designed SMT model is able to recover the state bit only if all of input classes are within the error tolerance 4 in key-stream generation phase, i.e., the predicted HW class should lie in the range $[\text{HW}_{org} - 4, \text{HW}_{org} + 4]$ (see Tables 5a, 5b and Section 5.1.1), where HW_{org} refers to the original HW class. Here, we focus on error tolerance 3 as SMT solution time is less but 0.216% of HW classes are incorrectly predicted. In order to correct those anomalies, we have designed an MILP model.

In order to show that our attack is generic, we first describe the MILP model on a general FSRs based stream cipher under HW model. Without loss of generality, we can assume the FSRs involved in the cipher are left oriented. As shown in Figure 3(a), assume the cipher has n registers (R_i, for $i \in \{0, 1, \ldots, n-1\}$) and their elements are shifted to their left position during update. The blue arrow represents the incoming and outgoing bits of each register and black arrow shows that bits are shifted in left direction during update. Let us divide the internal state into a sequence of consecutive blocks (B_i, for $i \in \{0, 1, \ldots, m-1\}$), each of length same as that of the microcontroller (8/16/32) used, see Figure 3(b). Dashed black arrow represents that either the last bit of block is shifted to its immediate left block or it is discarded.



(a) General structure of a stream cipher based on FSRs



(b) Consecutive blocks of internal states

Figure 3: Internal state and its block representation

The main idea of MILP modelling is to exploit the dependency relation of blocks shown in Figure 3(b) and keep track of incoming and outgoing bits. Using these relations we will search for a sequence of HWs which follows these relations and near to the predicted HWs sequence. Therefore the optimization function would be to minimise the distance of HWs sequence variables from predicted HWs sequence. It is to be noted that these relations have not been exploited by either ML or SMT. Now the algorithm to generate the constraints and objective function for the MILP instance from the block dependency relation is as follows.

Objective Function: Minimise the distance between sequence of HWs variables and sequence of predicted HWs.

Constraint I: Since there are n registers involved (see Figure 3(a)), the number of incoming bits for the internal state is equal to the number of outgoing bits (blue arrows), which is equal to the number of registers involved, i.e., n. It implies that the maximum change in HW of internal state for consecutive rounds is less than n.

Constraint II: For block B_i $(0 \leq i \leq m-1)$, assume that the number of incoming bit is in_i and the number of outgoing bit is out_i. Then the HW for block B_i after update either increases by at most in_i or decreases by at most out_i. See Figure 4, the blue line represents the incoming/outgoing bits from the register update function and black line represents the incoming/outgoing bits from its immediate left or right block.

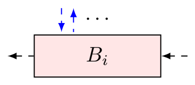


Figure 4: Representation of a single block

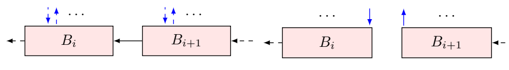
Constraint III: Here we explore the dependency relation for a pair of blocks of the type (a), see Figure 5(a), whereas we will discard the pairs of type (b), see Figure 5(b). Note in case (a) number of blue line could be zero. For the accepted pair of blocks we have the following two cases:

Case 1: Pair of the type as shown in Figure 6(a), where dashed arrow on block B_i shows either the last bit is fed to the block B_{i-1} (black) or being discarded (blue). Similarly for block B_{i+1} either the incoming bit at last position is coming from register update function or from the block B_{i+2}. In this case we can observe that the HW of the block B_{i+1} at round t will become the HW of the block B_i at $t + mc_{len}$ round, where mc_{len} is the length of microcontroller/block.

Case 2: Pair of the type shown in Figure 6(b). Let in_i (in_{i+1}) represents the number of incoming bits to the block B_i (B_{i+1}) respectively. Similarly, let out_i (out_{i+1}) represents the number of outgoing bits from the block B_i (B_{i+1}).

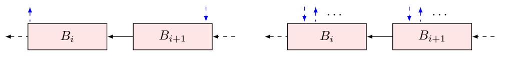
Now observe that if the HW of the block B_{i+1} increases by in_{i+1} (i.e., maximum increase) in the next round, then it implies that all outgoing bits are 0, which in turn implies that one of the incoming bits to the block B_i is 0 (see black arrow between B_{i+1} and B_i). It means that HW of the B_i can increase by at most $in_i - 1$ for that consecutive rounds. Similarly, if the HW of the block B_{i+1} decreases by out_{i+1} (i.e., maximum decrease) in the next round, then all outgoing bits from B_{i+1} are 1, which implies that one of the incoming bit to the block B_i is 1. Thus the HW of the block B_i can decrease by at most $out_i - 1$ in the next round.

As per the construction, sequence of original HW classes will satisfy constraints I, II and III. Through the MILP modelling we are trying to search for a sequence of classes which satisfies the above dependency relations, and are close to the sequence of predicted classes. So if there are few wrongly predicted classes, it will return a sequence which is more close to original class (because of Constraints I, II and III). Thereby we can formulate this as a minimisation problem. The MILP instance for the above is as given next.



(a) Accepted pair **(b)** Rejected pair

Figure 5: Types of block pair



(a) To another block **(b)** Affecting its previous block

Figure 6: Transfer of bits in a block

The following notations are used subsequently. Let N be the number of rounds, then for $0 \leq i \leq N-1$ and $0 \leq j \leq m-1$:

1. n is the number of registers in the stream cipher,
2. mc_{len} is the size of microcontroller used,
3. $\text{HW}_{var}[i,j]$ = MILP variable for HW of block j at round i,
4. $\text{HW}_p[i,j]$ = predicted HW of block j at round i,
5. $\text{HW}_{org}[i,j]$ = Original HW of block j at round i,
6. in_j represents the number of incoming bits to Block B_j,
7. out_j represents the number of outgoing bits from Block B_j.

Let $D_1 = \{0,\dots,N-1\}$ and $D_2 = \{0,\dots,m-1\}$. Then the mathematical representation of aforesaid MILP model is described subsequently.

Objective Function: Minimise $\sum_{i\in D_1, j\in D_2} |\,\text{HW}_{var}[i,j] - \text{HW}_p[i,j]\,|$

Type of Constraints: For $i \in D_1 \setminus (N-1)$ and $j \in D_2$, do the following.

Constraint I: $\left|\left(\sum_{j\in D_2} (\text{HW}_{var}[i,j] - \text{HW}_{var}[i+1,j])\right)\right| \leq n$

Constraint II: $-out_j \leq \text{HW}_{var}[i+1,j] - \text{HW}_{var}[i,j] \leq in_j$

Constraint III: For $j \in D_2 \setminus \{0\}$, such that the pair of blocks (B_{j-1}, B_j) is in the accepted pair category (see Figure 5(a)):

- **Case 1:** If the pair (B_{j-1}, B_j) is of the type as shown in Figure 6(a), then, $\text{HW}_{var}[i+mc_{len}, j-1] = \text{HW}_{var}[i,j]$
- **Case 2:** $(\text{HW}_{var}[i+1,j] - \text{HW}_{var}[i,j]) = in_j \implies (\text{HW}_{var}[i+1,j-1] - \text{HW}_{var}[i,j-1]) \leq in_{j-1} - 1$ and $(\text{HW}_{var}[i+1,j] - \text{HW}_{var}[i,j]) = -out_j \implies (\text{HW}_{var}[i+1,j-1] - \text{HW}_{var}[i,j-1]) \geq -(out_{j-1} - 1)$

In this system of MILP constraints, Constraint III comprises constraints of the form Condition 1 $\implies$ Condition 2. We can not model this implication relation in Gurobi directly. However, we can model this by converting it to the indicator constraint[13], which is of the form $(z = n) \implies$ Condition; for $n \in \{0,1\}$. Therefore the above conditional statement can be converted into a set of indicator constraints as follows: $(z = 1) \implies$ Condition 2; $(z = 0) \implies \neg$ (Condition 1) where $\neg$ denotes negation and z is a dummy binary variable.

Coming back to `TRIVIUM` MILP modelling, first note that `TRIVIUM` consists of right shift registers whereas the above mentioned model is on left shift register. Constraint I and II will remain same for both left and right FSRs based cipher, whereas in Constraint III, $j-1$ will be replaced by $j+1$. Since `TRIVIUM` comprises of 3 registers, we can replace n by 3 in Constraint I. For Constraint II, observe from Figure 7 that for any block, the number of incoming bits is same as the number of outgoing bits i.e., $in_j = out_j$, $0 \leq j \leq 8$. Therefore, $in_j = 1 = out_j \ \forall\ j \in \{0,1,3,4,6,7,8\}$, whereas $in_j = 2 = out_j \ \forall j \in \{2,5\}$.

[13]https://www.gurobi.com/documentation/9.1/refman/py_model_agc_indicator.html.

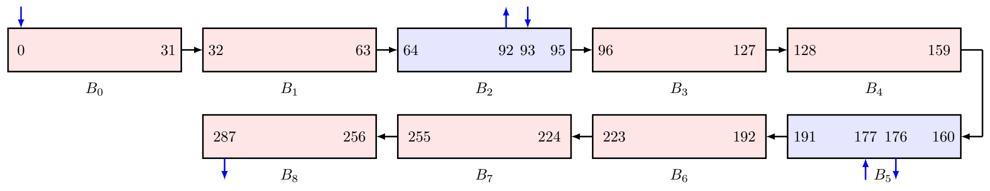


Figure 7: State registers of `TRIVIUM` (split into blocks of 32 bits)

Thus we can form Constraint II. In `TRIVIUM`, all consecutive pair of blocks comes under the category of accepted pair block (see Figure 5(a)). Therefore, Case 2 of Constraint III holds for all consecutive pair of blocks. For Case 1 of Constraint III, the block pairs $(B_0, B_1), (B_3, B_4), (B_6, B_7)$ and (B_7, B_8) are the right candidates.

In Table 4, we note down the results as follows. We conducted experiments on a simulated cipher, where the noises are injected in the same distribution as that of ML output (MLP-II in Table 3). We collect 1000 such random samples, where each sample is a sequence of noisy HW classes, where the noises follow the MLP-II distribution. Then we feed it to the MILP solver. To test its success probability, we note down how many experiments successfully corrected all of the predicted classes with tolerance 3. The percentage of this with respect to the total number of experiments is indicated in the third column.

As can be seen from Table 4, solution for Constraint I + II + III can be obtained in a matter of seconds. We can deduce from the table that success rate varies with the number of rounds. The best success rate is attained at round 110, i.e., 97.6. This means, out of 1000 independent experiments (where input to each experiment is a sequence of $110 \times 9 = 990$ classes), the model successfully corrects the input sequence in 976 experiments approximately. Suppose, we will repeat the same experiment independently for n times, then the probability that we will succeed at least once is, $1 - (1 - 0.976)^n \approx 0.999424$ (for $n = 2$), with trace information for 110 rounds. Thus we can conclude that we can correct sequence of predicted HW classes in 1 or 2 trial(s) with a very high probability, when traces information for 110 rounds are available.

All of the computations reported here are done on the MILP solver Gurobi-9.1.2 with Python-3.8.12, on an Intel Xeon E5-2670 v3 @ 2.30GHz CPU, running on a 64-bit Ubuntu-20.04 operating system. It is to be noted that with increase in the number of rounds, the solution time taken by Gurobi also increases. So if we needed to correct classes for higher number of rounds and solution time becomes high, then we can drop one or two constraints to reduce the solution time. Success probability might differ as per the types of constraints. Also Gurobi's performance varies with the system configuration and number of threads. In our experiment increasing the number of threads also increases solution time as well as memory, so we carried out our experiment using one thread.

Table 4: Success rate of MILP model for `TRIVIUM`

Constraint Type	#Rounds	Success Rate	Solution Time Mean (sec.)	Solution Time S.D. (sec.)
I + II + III	110	97.6	3.36	1.37
	130	95.9	4.84	0.98
	150	96.8	6.38	0.77
	170	94.6	7.86	1.18
	180	94.3	8.61	1.01
	200	93.3	10.53	0.83
	300	91.9	22.99	0.70

Tolerance = 3; Number of Threads used = 1

4.3.3 Part III: SMT to Solve for Unknown State/Key

Up to this point, we presume that the attacker has side channel traces in possession, obtains ML predicted sequence of classes (see Section 4.3.1). Then the attacker uses the MILP model to correct the sequence of predicted classes (refer to Section 4.3.2). Since the leakage data obtained is erroneous and it is not clear how to model arithmetic addition (for HW/HD equation), logical operation (for key-stream and state update function) and erroneous data in SAT simultaneously in a single system of equations. So we tried to convert the whole system of equations into a system of modular equations as modular operation is well supported through the data structure `BitVec`[14] in the SMT solver Z3. Instead of forming direct large degree equations, which is difficult to form and simplify, we form more smaller degree equations with the help of dummy variables, which work as follows: equate the dummy variable with the register update function and update the internal state with that dummy variable as the incoming bit (the concept of dummy variables is used before, see [Bar09] for an explanation or [BMS15] for another usage). Algorithm 1 is a generic way to form the SMT instances. For the mathematical formulation of SMT, assume that the MILP solver returns a HWs sequence, HW_p, such that $HW_p[i][j]-\epsilon \leq HW_{org}[i][j] \leq HW_p[i][j]+\epsilon$ for $i \in \{0,1,\ldots,N-1\}$, $j \in \{0,1,\ldots,8\}$ and some fixed $\epsilon > 0$, where N is the number of rounds and HW_{org} refers to the original HW class.

Now HW_p, key-stream Z, microcontroller size, tolerance ϵ and structure of the cipher are the inputs to Algorithm 1. In Step 1, $Nblock$ represents the number of consecutive blocks for the internal state, each of size mc_{len}. Since we are using modular operation, we have to define modulus large enough to accommodate maximum HW of any block, which is equal to the size of microcontroller, mc_{len}. So, we use the modulus of $2^{\lfloor \log_2(mc_{len}) \rfloor+1}$ operation [15] instead of arithmetic addition for HW, whereas for key-stream and update function we use logical operation. Thus, define $bitlen = \lfloor \log_2(mc_{len}) \rfloor + 1$ (Step 2). Initialise the solver m. Define state variables S_{var} of size $state_{len}$, each of the structure `BitVec(·,bitlen)` (Step 4). Similarly, define dummy variables of the same data structure for each register and for each round. For each such defined variables, put the constraints that its value is either 0 or 1, see Step 8. Now for each round, form the key-stream equations (Step 10) and HW class equation (Step 13) and feed it to the solver. During the state update operation, update the state normally (Step 15), extract update equations (from updated positions), equate it to dummy variables (Step 18) and put dummy variables at the updated positions (Step 19). Feed all the constraints to the SMT solver to solve. If all of the HW classes predictions are within tolerance ϵ and the solver returns a solution in a feasible time, then verify it (Step 26). If not verified, run SMT again by increasing the number of rounds and predicted sequence length. If it returns inconsistent, that means at least one predicted HW class fall outside the tolerance ϵ. For this case, follow the recovery procedure given in Section 3.2.4.

In case of `TRIVIUM`, we can proceed with the generic Algorithm 1 by using the parameter $state_{len} = 288, Nblock = 9, r = 3$, size of each register and their update position in the internal state. We carried out the SMT experiment using a simulated cipher, where we inject noise randomly (i.e., uniform distribution) to the original HW/HD information within the given tolerance class. The results of the experiments are shown in Section 5.

4.3.4 Part IV: Putting It All Together

For the attack during the pseudo-random generation phase of `TRIVIUM` (see Section 5.1.1 for the relevant results), note that our framework is able to work in a single key/IV setting.

[14]`BitVec(x,n)` It defines an n-bit variable x. We can perform modulo 2^n and logical operation simultaneously on this data structure.

[15]For example, if $mc_{len} = 32$ we can work in modulus 33, but since `BitVec` supports modulus operation only in a power of two, we have to model it in modulus $64 = 2^6 = 2^{\lfloor \log_2(mc_{len}) \rfloor+1}$.

Algorithm 1 SMT modelling for SCA (from HW data)

```
Input:
    i. Number of rounds, $N$
   ii. Internal state size, $state_{len}$
  iii. State registers $R_1, R_2, \dots, R_r$; with $|R_i| = r_i \forall i$
   iv. Position to update in the state, $pos$
    v. Microcontroller word size, $mc_{len}$
   vi. Tolerance, $\epsilon \geq 0$
  vii. An array of size $N$, $Z$, that stores the key-stream
 viii. Predicted classes $HW_p$ of size $N \times Nblock$
                                                                  ▷ Create Model
 1: $Nblock \leftarrow \lceil state_{len} \div mc_{len} \rceil$                              ▷ Number of blocks
 2: $bitlen \leftarrow \lfloor \log_2(mc_{len}) \rfloor + 1$                          ▷ $bitlen \geq \lceil \log_2 \max(\text{HW}) \rceil$
 3: $m \leftarrow$ Initialize Z3 solver
 4: Define $state_{len}$ number of BitVec variables for the internal state $S_{var}$, each of length $bitlen$-bit.
 5: $S \leftarrow S_{var}$
 6: Define dummy variables $R_i^{dum}$, $(1 \leq i \leq r)$ as a $bitlen$-bit BitVec variable each is an array of size $N-1$
                                                                  ▷ Generate constraints
 7: $var \leftarrow S_{var} + R_1^{dum} + R_2^{dum} + \dots + R_r^{dum}$
 8: Insert ULE($var[i], 1$) for $0 \leq i \leq |var|-1$ to the solver $m$        ▷ ULE : Unsigned $\leq$
 9: for $i \leftarrow \{0, 1, \dots, N-1\}$ do
10:     Insert key-stream($S$) $= Z[i]$ to the solver $m$                     ▷ (Optional)
11:     for $j \leftarrow \{0, 1, \dots, Nblock-1\}$ do
12:         $HW_{equ}[i][j] \leftarrow S[mc_{len} \times j] + \dots + S[mc_{len} \times j + mc_{len} - 1]$
13:         Insert $(|HW_{equ}[i][j] - HW_p[i][j]| \leq \epsilon)$ to $m$
14:     end for
15:     $S \leftarrow$ update($S$)                                      ▷ State update function
16:     if $i \neq N-1$ then
17:         for $j \leftarrow \{1, \dots, r\}$ do
18:             Insert $R_j^{dum}[i] = S[pos[j]]$ to $m$                    ▷ Use dummy variables
19:             $S[pos[j]] \leftarrow R_j^{dum}[i]$
20:         end for
21:     end if
22: end for
23: if $m$ is SAT then                                            ▷ Check SAT or UNSAT
24:     Print $m$                                                ▷ Print model information
25: end if
26: Verify if solution is correct with key-stream                        ▷ Sanity check
```

Also, attacking this phase is considerably more complex than attacking the initialisation phase, the relevant results on which can be found in Section 5.1.2.

The same key/IV setting is set for the attack during the initialisation phase (results are given in Section 5.1.2). In this case, since only 80 bits are now unknown (down from 288 in case of PRGA attack), the SMT solution time is drastically improved, even for higher tolerance, thereby cutting of necessity of the correction by the MILP model (Section 3.2.3).

Note that Algorithm 1 employs constraints found in the forward direction only. We can further utilise the backward equations, as the state update of TRIVIUM is invertible. By doing so, we have more equations with less degree monomials of dummy variables, which makes it easier to recover those variables. This idea is used only in the attack during the key-stream generation phase, whereas for the initialisation phase only forward equations are used as we start from round 0 to make use of IV. We optimize the Algorithm 1 further, by removing the condition 8, and defining all variables of the structure BitVec$(\cdot, 1)$ i.e., 1-bit variable. For key-stream and update equation we can proceed as given in the algorithm, whereas for HW equation we extend the variable size by appending zeroes using the function `ZeroExt(bitlen-1,·)`[16]. This is to perform modulus 2^{bitlen} operation on HW constraints.

The solution for SMT instances are carried out with Z3/Python 3.8.5. The CPU used for the computation is Intel Xeon CPU E5-2670 v3 @ 2.30GHz running a 64-bit Ubuntu-20.04.

[16]`ZeroExt(n,a)`: Returns a bit-vector expression with `n` extra zero-bits.

5 Results on TRIVIUM

5.1 Hamming Weight (Software) Model

5.1.1 Pseudo-random Phase

Table 5 shows the solution time taken by the SMT solver to produce unique solution under HW/8, HW/16 and HW/32 models and with varying error tolerance (i.e., its input is a sequence of predicted HW classes within error tolerance ϵ). In Table 5a, the key-stream equations are not included in the modelling (only update and HW equations are used), whereas Table 5(b) shows result when key-stream equations are included. Number of rounds is initially selected to be near the half of the number of original variables (i.e., 110). If the solution time is within few seconds and no multiple solutions is observed; then we increase error tolerance by 1, otherwise we increase/decrease the number of rounds by 20–30 to check if solution time decreases and unique solution exists. If we take number of rounds too low, then we generally have multiple solutions[17].

In Table 5(a), under HW/8 model (recall the description of HW/8, HW/16 and HW/32 from Section 4.1) we can recover state bit up to the error tolerance 3, which is almost half of the corresponding word size of the microcontroller. Beyond error tolerance 3, Z3 returns multiple solutions even for higher rounds, and also takes longer time to solve. For HW/16 model, we could find solution for up to tolerance 4 in about 1–2 hours. For the 32-bit microcontroller, the results are as described next. As per Tables 5(a) and 5(b), we have the SMT solution time up to the error tolerance $\epsilon = 4$. Table 4 shows results for the success rate of MILP modelling only for tolerance 3, as in this case we have more results and less SMT time. But for lower tolerance more predicted HWs are wrong (i.e., lies outside the expected tolerance limit), which in turn makes SMT system of equations inconsistent (as positions for wrong classes are unknown). Aforementioned MILP model can correct these wrong HWs and return a sequence of HW classes within the tolerance 3 with a very high success rate (see Table 4). We now describe the total solution time with respect to tolerance 3 (as we have more solutions for this bound of tolerance, than that of 4). For 150 rounds, the success probability is 0.968 (see Table 4), Gurobi takes 6.38 seconds to solve and the SMT solution time is 49975.49 second (see Table 5(a)). So the total required time is 49981.87 seconds with a success probability of 0.968. The best result can be obtained for 170 rounds (see Table 5(b)) with solution time $28755.36 + 7.86 = 28763.22$ seconds with probability 0.946 (only 1 sample is available). Similarly for 180 rounds, the solution time is $36288.8 + 8.61 = 36297.41$ seconds with success probability of 0.943.

Therefore, to conclude, we can recover the state bit of TRIVIUM in 28763.22 seconds for 170 rounds (with key-stream equations) with probability 0.946. The SMT solution time is 36297.41 seconds for 180 rounds (without key-stream equations) with a success rate of 0.943. We ignore the overhead time taken to form the SMT and MILP instances, as it is within a few seconds. Note that, the success probability is less than 1. If the SMT instance returns inconsistent in the first trial, then for next trial we can target the state at different/same round during key-stream phase under same key/IV. Therefore, we do not need to change IV or rerun the cipher for another trace.

For the computation of the success probability in n trails, note the following. Let p be the success probability in first trial. Then the probability that we get success at least once in n trials is $1 - (1 - p)^n$. The result above with best probability is 0.968 (150 rounds). Therefore, on 2 trials we will succeed at least once with a probability of 0.9989.

[17]If multiple solutions are found, we increase the number of rounds until the unique solution is found.

Table 5: Results on `TRIVIUM` in pseudo-random phase

(a) Without key-stream information

Leakage Model	Tolerance	# Rounds	Trials	Mean (sec.)	S.D. (sec.)
HW/8	1	110	20	1.16	0.05
	2	110	20	1.37	0.07
	3	110	20	1.58	0.13
HW/16	1	110	20	3.91	0.94
	2	110	20	7.38	2.18
	3	110	20	15.41	6.35
	4	110	20	91.40	187.50
		130	20	40.01	22.85
		150	20	39.89	18.62
HW/32	1	110	20	239.74	192.64
	2	110	20	7975.88	9277.67
		130	20	4764.87	4489.14
	3	130	6	122582.24	65397.23
		150	6	49975.49	31924.09
		180	5	36288.8	27153.63
	4	130	1	475778.30	–
		150	3	49445.14	38190.05
		170	2	226005.79	71432.18

(b) With key-stream information

Leakage Model	Tolerance	# Rounds	Trials	Mean (sec.)	S.D. (sec.)
HW/32	0	110	20	5.42	1.255
	1	70	20	1309.19	1144.36
		90	20	821.76	1444.34
		110	20	254.36	195.43
	2	70	1	3408.72	–
		90	8	17404.32	27533.08
		110	16	10628.26	13962.97
		130	20	1819.21	1558.38
	3	110	1	140523.60	–
		130	3	44911.09	31619.74
		170	1	28755.36	–
	4	130	1	76797.41	–
		170	1	12582.60	–

5.1.2 Initialisation Phase

In the initialisation phase of `TRIVIUM`, the number of unknown variables reduces from 288 to 80 as only secret key bits (80 bits) are unknown. This enable us to solve the SMT instance for higher tolerance, as we describe next. While the results in our paper are all given with a single IV, we like to note that our SMT model can readily deal with multiple IVs. Further, since for each IV the system gets some new information, the solution time will likely be less and we can solve SMT instances for higher error tolerance.

Table 6: Results on `TRIVIUM` in the initialisation phase

Leakage Model	Tolerance	# Rounds	Trials	Mean (sec.)	S.D. (sec.)
HW/8	4	150	20	1.63	0.28
HW/16	5	140	20	2.12	0.23
	6	140	20	2.46	0.698
	7	140	20	4.45	2.13
	8	200	20	153.01	223.71
HW/32	3	140	20	4.93	1.55
	4	140	20	8.58	6.90
	5	140	20	10.39	11.39
		160	20	11.64	6.44
	6	140	20	27.07	34.97
	7	170	20	79.49	121.89
	8	160	20	211.89	636.58
	9	160	20	162.08	186.37
	10	160	20	585.56	1484.28
	11	160	20	1018.86	1629.53
	12	160	20	4560.34	5749.65
	13	160	11	3646.35	3632.99
	14	200	2	18379.19	5143.47
		300	2	5566.90	4175.17
		280	3	1460.03	1234.7
	15	280	1	5859.60	–

Table 6 shows solution time for HW/8, HW/16 and HW/32 in the initialisation phase. For a given tolerance, initially we select the number of rounds as 140 (as for 130–140 rounds and tolerance 3–4 in key-stream generation phase, the SMT solution time is feasible) see Tables 5a and 5b. Thereafter, if multiple solutions are observed then we increase the number of rounds by 10 and solve. Notice under the HW/8 and HW/16 scenarios, we reach at the optimal point at the half of the word size of the microcontroller[18].

For the HW/32 model during the initialisation phase, we can solve up to error tolerance 15 (almost near to the optimal point). However, as per Table 3, HW class can be predicted with 100% accuracy for error tolerance 7. Thus, we can create the SMT instance with any tolerance beyond 6 and solution time can be found with probability 1. Therefore, we do not use the MILP based correction here. In this case, we can achieve the best solution time of 79.49 seconds (170 rounds) with probability 1.

On the other hand, as given in Table 3, the accuracy for tolerance 4 is 0.99965. Therefore with HW information for 140 rounds (i.e., HW prediction of $140 \times 9 = 1260$ blocks), on an average $140 \times 9 \times 0.999655 \approx 1260$ HW data are within error tolerance of 4. Thus, we also solve the SMT instance for tolerance 4 to obtain a solution within 8.58 seconds. If it returns inconsistent, we increase the tolerance and solve the equations (refer to Section 3.2.4 for more details on the recovery procedure).

5.2 Hamming Distance (Hardware) Model

On top of the MILP model for HW (as detailed in Section 4.3.2), the following observations are used for MILP modelling for HD leakage model. Note that, the HD of a register changes by $\{-1, 0, 1\}$ during state update. Therefore, total changes for an n-register cipher changes by $\{-n, -n+1, \ldots, n-1, n\}$. For SMT modelling, note that Algorithm 1 also holds for HD model when the HW-constraints are replaced with the HD-constraints under modulus $2^{\lfloor \log_2(state_{len}) \rfloor + 1}$ (as the maximum possible HD is equal to the state size, $state_{len}$).

5.2.1 Initialisation Phase

In Table 7, we present the results for `TRIVIUM` under HD model in the initialisation phase. Let the internal state at the targeted round t be denoted as $S_t = [s_0, s_1, \ldots, s_{287}]$, i.e., a tuple of unknown variables. The last column indicates the location for the bits which are guessed.In the same column, $s_i \cdots s_j$ for $0 \le i \le j \le 287$, denotes that all variables in the range $s_i, s_{i+1}, s_{i+2}, \ldots, s_j$ are guessed.

Table 7: Results for `TRIVIUM` on HD model

Phase	Tolerance	#Guess	#Rounds	Trials	Mean (sec.)	SD (sec.)	Guessed Bit(s)
Initialisation	0	0	90	20	238.06	418.42	
	0	0	100	20	144.84	443.75	–
	0	0	150	20	121.64	197.83	–
	1	0	200	4	71887.63	58920.45	–
	1	5	200	5	22401.33	24507.83	$s_{65} \cdots s_{69}$
	1	10	150	5	1802.77	1111.69	$s_{60} \cdots s_{69}$
	1	10	150	5	11126.37	17273.22	$s_{56} \cdots s_{65}$
	1	20	150	5	673.06	683.08	$s_{60} \cdots s_{79}$
	1	20	200	5	442.68	528.62	$s_{60} \cdots s_{79}$
	2	10	200	3	205110.06	112955.45	$s_{56} \cdots s_{65}$
	2	30	200	5	223.98	126.74	$s_{36} \cdots s_{65}$
	3	20	200	3	3603.99	3131.14	$s_{46} \cdots s_{65}$
	3	20	220	3	62844.57	84668.85	$s_{46} \cdots s_{65}$
	3	30	200	5	1012.02	1198.11	$s_{40} \cdots s_{69}$
	3	30	200	5	2133.05	3129.14	$s_{36} \cdots s_{65}$

We tried up to tolerance 3 only, but it can also be extended for higher tolerance. The guessed bits are chosen consecutively near the variables which are involved in state update function. It is selected consecutively because during update the variables inside the register are shifted to its right position, i.e., variable on the immediate right side of the tap[19] position moves to the tap position in the next round. From Table 7, we can see that up to tolerance 1 we can recover state bits without any guess. However for higher tolerance 2, we

[18] 'Optimal' here means, beyond that tolerance we get multiple solutions even at higher rounds.

[19] By tap position, we refer to the position in the state which is used the key-stream/update function.

need at least 10 guessed bits and for tolerance 3, we need at least 20 guessed bits. All of that can be done in practical time. For guessing, we have to go through all the possibilities and try SMT algorithm for each guess. However, the SMT time would be lower for a wrong guess, i.e., it will return inconsistent for a wrong guess faster than consistent result for the right guess.

5.2.2 Pseudo-random Phase

Regarding the pseudo-random phase, the system of constraints here has 288 unknown variables which is quite high as compared to the 80 variables in the initialisation phase. We do not get any result till 2 days. However, with a guess of 140 bits we get the result, but doing so exceeds the complexity of exhaustive search on the key (which is only of 80-bits). A probable reason behind this is that we have a large number of unknown variables and the key-stream in `TRIVIUM` involves only 6 bits which does not carry enough information. The higher effort required to attack pseudo-random phase as compared to initialization phase can also further motivate use of levelled implementations [BBC+20], where resource friendly countermeasures may be considered for pseudo-random phase, while initialization must be well protected.

6 Success Probability with respect to varying SNR

We test our framework (in pseudo-random phase) with different SNRs. We add Gaussian noise with 0 mean and different standard deviations to the previously used traces to simulate low SNR settings. The parameters used are as follows: 2^{20} data-set size, 50 epochs, MLP with two hidden layers of 128 neurons each, a 62.5/12.5/25 training/validation/testing split and a batch size of 64. We used the same MLP architecture that was used for the original experiment. The accuracy of the output is then fed to MILP, which runs on a simulated cipher and noises are injected as per the ML accuracy. We tested MILP experiments for 130 and 150 rounds as SMT solver needs information of either 130 rounds or 150 rounds for state recovery under error tolerance 3 (see Table 5(a)). The MILP experiments are carried out using 1000 random trials. Thus the success probability of MILP (success probability of the whole experiment) with respect to varying standard deviation is shown in Table 8. From the table, we can see that beyond SNR 1.12124, the success probability drops to 0. Thus, SNR 1.12124 is the threshold for our framework with the current experiment setting. Alternatively, one can also optimise ML architecture to boost accuracy at lower SNR (where possible), to perform key recovery at lower SNR.

Table 8: Success Probability with respect to SNR

Noise Standard Deviation	SNR	Testing Accuracy for different error tolerance (ϵ)				Success Probability w.r.t. different number of rounds, N	
		0	1	2	3	N = 130	N = 150
3.5	2.34751	0.25324	0.66331	0.88699	0.97158	0.54	0.511
3.75	2.24475	0.26465	0.68409	0.90211	0.97717	0.614	0.614
4.3	2.05877	0.23549	0.62463	0.85628	0.95744	0.36	0.39
4.5	1.99083	0.23652	0.62912	0.86226	0.95995	0.331	0.323
5.0	1.83929	0.23549	0.62475	0.85615	0.95715	0.347	0.307
6.0	1.54966	0.20732	0.56019	0.79589	0.92121	0.094	0.066
6.2	1.50017	0.19534	0.53439	0.77478	0.90953	0.06	0.038
6.6	1.42157	0.19354	0.52944	0.76958	0.90731	0.044	0.033
7.5	1.21922	0.18766	0.51646	0.75444	0.89274	0.02	0.016
8.0	1.12124	0.18295	0.50795	0.74527	0.88698	0.016	0.016
9.0	0.95195	0.15937	0.44838	0.67646	0.83112	0.001	0.001
10.0	0.82012	0.14931	0.42224	0.6423	0.79988	0.0	0.0

7 Conclusion

In this paper, we show a pragmatic framework that recovers the state/key from stream ciphers and related constructions from the side channel information (power or EM). Our framework is able to attack the initialisation phase (i.e., before the cipher reaches its pseudo-random phase) and, more importantly even after the cipher reaches its pseudo-random phase to produce key-stream. The efficacy of our framework is shown through the EM leakage from a 32-bit software platform on the high-profile stream cipher `TRIVIUM` in Section 5.1. To add the cherry-on-top, we also show how our model can work with the hardware leakage in Section 5.2.

Our framework is based on profiling of side channel leakage, i.e., it works at offline and online stages. In the offline stage, the attacker collects information about the side channel leakage to build a profile, which is then used during the online stage to find out secret information. The analysis of the offline stage starts with an ML model (Section 4.3.1) where we use an MLP with only 2 hidden layers. Since our target software platform has 33 HW classes, our first intuition is to see the accuracy by the model for each of the individual 33 classes. In the process though, we discover that it is hard to go beyond 40% accuracy, even with deeper networks. Thus, we introduce the concept of tolerance (denoted by ϵ), where we broaden the scope for correct prediction from the ML model. For a given ϵ and the correct class c, we count the prediction as correct with tolerance ϵ if the predicted class lies within the $2\epsilon + 1$ neighbourhood of c, i.e., $\{c - \epsilon, c - \epsilon + 1, \ldots, c, \ldots, c + \epsilon - 1, c + \epsilon\}$. This allows for a higher accuracy from the ML model as ϵ increases. The offline stage ends with an SMT solving (Section 4.3.3), that returns the state/key of the cipher. However, increasing ϵ comes with the catch that the solution time taken by the SMT solver increases. Thus, it becomes important to find the proper balance where the ML accuracy is considerably high and the solution time taken by the SMT solver is reasonably low. With our experimental data from the 32-bit ARM Cortex-M3, we settle at $\epsilon = 3$ where the ML accuracy is about 99.7%, and the SMT solution time with our target cipher `TRIVIUM` is about few hours. Even though the ML accuracy is quite high, still certain wrong (i.e., outside the desired ϵ limit) predictions will pass through. One such constraint, again, will make the system inconsistent. To mitigate that, we introduce an intermediary between ML and SMT, which is based on MILP (Section 4.3.2). This MILP model takes the sequence of predictions from the ML model, and transforms (with a high probability) into another sequence where all the predictions lie within the desired ϵ level.

Additionally, to test the limits of our approach for varying SNRs, we added Gaussian noise to the previously measured traces. We observed that at SNR of ≈ 2 and ≈ 1.1, the ML accuracy falls to 0.95744 and 0.88698 (for error tolerance 3), respectively. This decrease in ML accuracy degrades the success probability of key recovery to roughly 0.36 (for SNR=2) and 0.016 (for SNR=1.1). We note that one can explore different pre-processing techniques and can optimise the ML architecture to mitigate the effect of low SNR on the success probability.

To conclude, we list the following problems that may be of interest as follow-up works:

- **Analytical approach for less than perfect accuracy:** Our approach requires a strict condition that the accuracy of the HW/HD prediction has to be 100%, otherwise the SMT instance will become inconsistent. However, it may be possible to come up with an analytical approach, such as a *Hidden Markov Model* (HMM), that can work (with a high probability) when the accuracy is lower than 100%. This can take some load from the ML module (Section 4.3.1) and can potentially remove the MILP module (Section 4.3.2).
- **Improvement of ML:** First, while we attempt to find an efficient ML model, this is not the focus of this research, and can likely be improved. For instance, one may experiment with other types of ML models, such as *Convolutional Neural Networks* (CNN), *Recurrent Neural Networks* (RNN), or *Long Short-Term Memory (LSTM)*

networks. The electromagnetic traces can be treated as time series data and exploring the feasibility of RNN/LSTM would be an interesting direction of study. Second, data pre-processing techniques such as normalisation, scaling, and feature selection can be used for a potential performance gain and improvement of the training stability. For instance, instead of feeding the complete trace to the ML model, we can identify points of interest and only use those to reduce the complexity. Third, the ML model does not take into account the specifics of the cipher (those specifics are utilised under the MILP module in Section 4.3.2). This can be considered in a more general setting, ultimately cutting-off the MILP module.

- **Form of leakage function:** Extension of our framework for polynomial leakage function [GSP13] and weighted HW/HD leakage function [DPRS11] instead of HW (software) or HD (hardware) can be considered in the future. These leakage functions will probably have an impact on our framework, as in our case, information on a linear equation gives better results than the information on a general polynomial equation. For a weighted leakage function, since the current SMT model works on modular operations, for a negative weighted leakage function the SMT model needs to be improved. Additionally, it also depends upon SNR of the side-channel measurements.

References

[Bar09] Gregory Bard. *Algebraic cryptanalysis.* Springer Science & Business Media, 2009. 180

[BBC$^+$20] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 369–400. Springer, 2020. 185

[BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. Give me 5 minutes: Attacking ASCAD with a single side-channel trace. *IACR Cryptol. ePrint Arch.*, page 817, 2021. 170

[BDGN13] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. NICV: normalized inter-class variance for detection of side-channel leakage. *IACR Cryptol. ePrint Arch.*, page 717, 2013. 174

[BMS15] Anubhab Baksi, Subhamoy Maitra, and Santanu Sarkar. An improved slide attack on Trivium. *Journal IPSI Transaction on Internet Research*, 2015. 180

[BS97] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In Burton S. Kaliski, editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer Berlin Heidelberg, 1997. 167

[CFGR12] Claude Carlet, Jean-Charles Faugère, Christopher Goyet, and Guénaël Renault. Analysis of the algebraic side channel attack. *J. Cryptogr. Eng.*, 2(1):45–62, 2012. 169

[CMM15] Abhishek Chakraborty, Bodhisatwa Mazumdar, and Debdeep Mukhopadhyay. Combined side-channel and fault analysis attack on protected grain family of stream ciphers. *IACR Cryptol. ePrint Arch.*, 2015:602, 2015. 169

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008. 169

[DCP08] Christophe De Canniere and Bart Preneel. Trivium. In *New stream cipher designs*, pages 244–266. Springer, 2008. 168, 171, 174

[DPRS11] Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *J. Cryptogr. Eng.*, 1(2):123–144, 2011. 187

[FGKV07] Wieland Fischer, Berndt M. Gammel, O. Kniffler, and Joachim Velten. Differential power analysis of stream ciphers. In Masayuki Abe, editor, *Topics in Cryptology - CT-RSA 2007, The Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, February 5-9, 2007, Proceedings*, volume 4377 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 2007. 169

[GBC$^+$08] Benedikt Gierlichs, Lejla Batina, Christophe Clavier, Thomas Eisenbarth, Aline Gouget, Helena Handschuh, Timo Kasper, Kerstin Lemke-Rust, Stefan Mangard, Amir Moradi, and Elisabeth Oswald. Susceptibility of estream candidates towards side channel analysis. sasc –the state of the art of stream ciphers. In *Workshop Record*, pages 123–150, 2008. 167, 169

[GGSB20] Qian Guo, Vincent Grosso, François-Xavier Standaert, and Olivier Bronchain. Modeling soft analytical side-channel attacks from a coding theory viewpoint. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):209–238, 2020. 170

[GSP13] Vincent Grosso, François-Xavier Standaert, and Emmanuel Prouff. Low entropy masking schemes, revisited. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 33–43. Springer, 2013. 187

[HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *Int. J. Wirel. Mob. Comput.*, 2(1):86–93, 2007. 169

[HKM17] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD - A lightweight stream cipher for power-constrained devices. *IACR Trans. Symmetric Cryptol.*, 2017(1):45–79, 2017. 174

[HLM$^+$20] Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against trivium and grain-128aead. In *EUROCRYPT 2020, Zagreb, Croatia, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 466–495. Springer, 2020. 174

[HYY$^+$10] Matthew Henricksen, Wun-She Yap, Chee Hoo Yian, Shinsaku Kiyomoto, and Toshiaki Tanaka. Side-channel analysis of the K2 stream cipher. In Ron Steinfeld and Philip Hawkes, editors, *Information Security and Privacy - 15th Australasian Conference, ACISP 2010, Sydney, Australia, July 5-7, 2010. Proceedings*, volume 6168 of *Lecture Notes in Computer Science*, pages 53–73. Springer, 2010. 169

[Jaf07] Joshua Jaffe. A first-order DPA attack against AES in counter mode with unknown initial counter. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2007. 169

[JBL19] Martin Jurecek, Jirí Bucek, and Róbert Lórencz. Side-channel attack on the a5/1 stream cipher. In *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pages 633–638. IEEE, 2019. 169

[KAA$^+$17] Asif Raza Kazmi, Mehreen Afzal, Muhammad Faisal Amjad, Haider Abbas, and Xiaodong Yang. Algebraic side channel attack on trivium and grain ciphers. *IEEE Access*, 5:23958–23968, 2017. 169

[KEI$^+$21] Hayato Kimura, Keita Emura, Takanori Isobe, Ryoma Ito, Kazuto Ogawa, and Toshihiro Ohigashi. Output prediction attacks on block ciphers using deep learning. Cryptology ePrint Archive, Report 2021/401, 2021. `https://eprint.iacr.org/2021/401`. 175

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. 167

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996. 167

[LH17] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. 175

[MBZ$^+$13] Mohamed Saied Emam Mohamed, Stanislav Bulygin, Michael Zohner, Annelie Heuser, Michael Walter, and Johannes Buchmann. Improved algebraic side-channel attack on AES. *J. Cryptogr. Eng.*, 3(3):139–156, 2013. 170

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007. 167, 174

[OKPW10] Yossef Oren, Mario Kirschbaum, Thomas Popp, and Avishai Wool. Algebraic side-channel analysis in the presence of errors. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2010. 170

[ORSW12] Yossef Oren, Mathieu Renauld, François-Xavier Standaert, and Avishai Wool. Algebraic side-channel attacks beyond the hamming weight leakage model. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 140–154. Springer, 2012. 169, 170

[Pee13] Eric Peeters. *Advanced DPA Theory and Practice: Towards the Security Limits of Secure Embedded Circuits*. Springer-Verlag New York, 1 edition, 2013. 167

[PHJ+17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 4095–4102, 2017. 175

[QGGL13] Bo Qu, Dawu Gu, Zheng Guo, and Junrong Liu. Differential power analysis of stream ciphers with lfsrs. *Comput. Math. Appl.*, 65(9):1291–1299, 2013. 169

[RO04] Christian Rechberger and Elisabeth Oswald. Stream ciphers and side-channel analysis. In *In ECRYPT Workshop, SASC-The State of the Art of Stream Ciphers*, pages 320–326. Citeseer, 2004. 167, 169

[RS09] Mathieu Renauld and François-Xavier Standaert. Algebraic side-channel attacks. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Information Security and Cryptology - 5th International Conference, Inscrypt 2009, Beijing, China, December 12-15, 2009. Revised Selected Papers*, volume 6151 of *Lecture Notes in Computer Science*, pages 393–410. Springer, 2009. 169

[RSV09] Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the AES: why time also matters in DPA. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2009. 169, 170

[RSV+11] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2011. 174

[SJB20] Siang Meng Sim, Dirmanto Jap, and Shivam Bhasin. Dapa: Differential analysis aided power attack on (non-) linear feedback shift registers (extended version). *IACR Cryptol. ePrint Arch.*, 2020:1241, 2020. 169, 174

[SMB17] Santanu Sarkar, Subhamoy Maitra, and Anubhab Baksi. Observing biases in the state: case studies with Trivium and Trivia-SC. *Design, Codes and Cryptography*, 82(1-2):351–375, 2017. 174

[Str09] Daehyun Strobel. Side channel analysis attacks on stream ciphers. *Masterarbeit Ruhr-Universität Bochum, Lehrstuhl Embedded Security*, 2009. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.182.1943&rep=rep1&type=pdf. 169

[TA15] Erica Tena-Sánchez and Antonio J. Acosta. DPA vulnerability analysis on trivium stream cipher using an optimized power model. In *2015 IEEE International Symposium on Circuits and Systems, ISCAS 2015, Lisbon, Portugal, May 24-27, 2015*, pages 1846–1849. IEEE, 2015. 169

[TSA15] Erica Tena-Sánchez and Antonio J Acosta. Optimized dpa attack on trivium stream cipher using correlation shape distinguishers. In *2015 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2015. 169

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014. 170

[ZWG$^+$11] Xinjie Zhao, Tao Wang, Shize Guo, Fan Zhang, Zhijie Shi, Huiying Liu, and Kehui Wu. Sat based error tolerant algebraic side-channel attacks. In *2011 Conference on Cryptographic Algorithms and Cryptographic Chips, CASC*, 2011. 170