Algebraic Side-Channel Analysis in the Presence of Errors

Yossef Oren[1] and Mario Kirschbaum[2] and Thomas Popp[2] and Avishai Wool[1]

[1] Computer and Network Security Lab, School of Electrical Engineering
Tel-Aviv University, Ramat Aviv 69978, Israel
{yos, yash}@eng.tau.ac.il
[2] Institute for Applied Information Processing and Communications
Graz University Of Technology, Inffeldgasse 16a, A-8010, Austria
{mario.kirschbaum,thomas.popp}@iaik.tugraz.at

Abstract. Measurement errors make power analysis attacks difficult to mount when only a single power trace is available: the statistical methods that make DPA attacks so successful are not applicable since they require many (typically thousands) of traces. Recently it was suggested by [18] to use algebraic methods for the single-trace scenario, converting the key recovery problem into a Boolean satisfiability (SAT) problem, then using a SAT solver. However, this approach is extremely sensitive to noise (allowing an error rate of well under 1% at most), and the question of its practicality remained open. In this work we show how a single-trace side-channel analysis problem can be transformed into a *pseudo-Boolean optimization* (PBOPT) problem, which takes errors into consideration. The PBOPT instance can then be solved using a suitable optimization problem solver. The PBOPT syntax provides for a more expressive input specification which allows a very natural representation of measurement errors. Most importantly, we show that using our approach we are able to mount successful and efficient single-trace attacks even in the presence of realistic error rates of 10%–20%. We call our new attack methodology *Tolerant Algebraic Side-Channel Analysis (TASCA)*. We show practical attacks on two real ciphers: Keeloq and AES.

Keywords: Algebraic attacks, power analysis, side-channel attacks, pseudo-Boolean optimization

1 Introduction

1.1 Background

Side-channel cryptanalysis has been an active field of research for the last 15 years. For the simplest devices, that are susceptible to Simple Power Analysis attacks (SPA) [12], the secret key can be read directly from the shape of a side-channel trace (power consumption, EM radiation, etc.). More commonly, the cryptanalyst needs to use differential (DPA) analysis [12,14]. DPA techniques

typically require multiple traces, often hundreds or more, to overcome the measurement noise via signal processing and statistical estimation techniques. Obtaining all these traces places a significant burden on the attacker, and it is quite interesting to discover ways to extract the secret key data from a *single trace* from devices that are not susceptible to SPA.

Recently it was suggested by [18] to separate the problem into two separate phases: the first phase is the *estimation phase*, where information is extracted from the power traces using signal processing techniques, while the second is the *key recovery phase*, where this information is processed to return cryptanalytically significant results. In particular, [18] uses *algebraic methods* for the key recovery phase, converting the problem into a Boolean satisfiability (SAT) problem, then using a SAT solver. Algebraic cryptanalytic attacks using external solvers were first explored by Massacci and Marraro in [16] in the context of conventional cryptanalysis. However, these attacks are difficult to apply directly to side-channel attacks, since the SAT representation of a cryptosystem and its side-channel measurements is extremely sensitive to noise — indeed [18] were only able to solve problems with an error rate well under 1%, which is much lower than realistic noise on a single trace. Side-channel analysis using standard solvers was also suggested by [17]. Our goal in this paper is to demonstrate a more promising algebraic cryptanalysis approach, based on Pseudo-Boolean optimization, which is able to withstand much higher error rates.

Other non-algebraic methods have also been suggested for dealing with single-trace power analysis in the presence of noise. A side-channel attack using the Viterbi iterative algorithm [20] for dealing with errors, first presented in the context of elliptic-curve operations in [11], is one example.

1.2 Causes of Errors in Side-Channel Information

The side-channel information emitted by a cryptographic device is an analog high-frequency signal that is measured with a suitable instrument. In case of the power consumption side-channel, the logic cells in the digital circuit draw power from the supply according to their state and activity. This instantaneous power consumption signal is measured with an oscilloscope. The measurement process includes an analog-to-digital conversion of the sampled values. On their way from the logic cell to the oscilloscope's digital output, the power values are influenced by all kinds of physical effects and other signals. These influences are commonly denoted as *noise.* This noise can cause *decoding errors* when trying to estimate the original power consumption of the logic cell.

The overall noise that is present in measured power traces can be divided into electronic noise, quantization noise, and switching noise. Electronic noise is present in every measurement in practice. It includes the noise that occurs in conductors (e.g. thermal noise) and semiconductors (e.g. generation-recombination noise). Furthermore, sources of electronic noise are the conducted and radiated emissions from all components that are part of the control and measurement setup and from external components that operate in the vicinity of the measured cryptographic device. These components include the supply unit that powers the

device and the oscilloscope. Another important source of electronic noise is the clock generator that supplies the digital circuit in the cryptographic device with the clock signal. Due to its typical rectangular shape, this signal contains high-frequency components that also influence the measured power values.

The digital oscilloscope contains another source of noise. The analog-to-digital conversion process that it performs introduces small errors in the measured values. The effect of these errors can be modeled as noise in the measured signal, commonly called *quantization noise*. The higher the resolution of the oscilloscope, the lower is the amount of quantization noise.

The third main type of noise is switching noise. Besides the power consumption of the logic cells we are interested in, typically also other cells contribute to the total power consumption value at a specific point in time. The power signals from these other cells are denoted as switching noise. The main parameters of the control and measurement setup that influence the amount of switching noise for a specific point in time are the bandwidth of the power measurement system and the clock frequency. The lower the bandwidth of the measurement path the more the distinct power consumption signals of individual logic cells get blurred together and the amount of switching noise increases. A higher clock frequency can also have such an effect[14].

1.3 Contributions

Our key observation is that a SAT representation does not offer a very convenient or efficient method to deal with errors in side-channel information. Instead, we suggest casting the problem in the more expressive language of non-linear *pseudo-Boolean optimization* (PBOPT).

A PBOPT representation offers several properties that are suitable for single-trace side-channel attacks in the presence of errors: (a) A side-channel measurement is typically the Hamming weight w of some hardware feature: this is naturally represented by equating a sum of state bits to the *integer* w – in contrast, representing integers in a SAT instance is quite awkward; (b) It is straight-forward to add variables representing error quantities to the side-channel equations; (c) Unlike a SAT, that is basically a decision ("yes/no") problem, a PBOPT instance includes an objective function, and the solver finds a solution that minimizes this objective.

Luckily, PBOPT offers more than a convenient representation formalism. Research on non-linear pseudo-Boolean equation solvers is a field which displays remarkable activity, and even has a highly-competitive yearly evaluation of solvers [15]. Thus, a PBOPT instance representing a single-trace side-channel attack with errors can actually be solved efficiently, leading to our new attack methodology: *Tolerant Algebraic Side-Channel Analysis (TASCA)*.

To demonstrate the viability of our TASCA approach, we mounted successful and efficient single-trace attacks, against real, fielded ciphers, even in the presence of realistic error rates of 10%–20%. We show a practical attack on the Keeloq system, and preliminary results on AES.

Organization: The next section describes the basics of algebraic side-channel attacks. Section 3 describes our new *Tolerant Algebraic Side-Channel Analysis (TASCA)* approach. Section 4 shows the effectiveness of TASCA against a power-simulated ASIC implementation of Keeloq, and Section 5 shows preliminary results against a power-simulated 8-bit microcontroller implementation of AES-128. Section 6 suggests some open problems. We conclude with Section 7.

2 Algebraic Side-Channel Attacks

2.1 General Structure of an Algebraic Attack

As stated in [18], the cryptanalytic problem needs to be transformed into a set of equations before being submitted to the equation solver. This equation set typically consists of a general description of the cryptographic algorithm, together with an assignment of any known inputs to the algorithm. If the equation set represents an algebraic side-channel attack, it will contain additional equations which describe the side-channel emanations of the system in addition to the standard known plaintexts and ciphertexts. Building on the results of [18], we can assume that an errorless description of the side-channel data will lead to successful key recovery. However, such an equation set is very sensitive to noise: a single errored side-channel measurement will create an equation set that is either unsatisfiable, or is satisfied by the wrong key.

The equations are presented to the solver using the solver's problem description language. The authors of [18] used a SAT solver which accepts its input in the form of conjugate normal form (CNF) SAT statements. As we shall see, we use the richer and more powerful pseudo-Boolean optimization representation.

2.2 Naïve Methods of Dealing with Errors

Assume that the vector z represents some side-channel information extracted from a certain cryptographic operation (for example Hamming weights or Hamming distances) under a certain key k_c, and that there exists some distance function $d(k, z)$ which indicates how likely a given vector z is to be the result of the operation under a certain key k. As noted in the introduction, the raw side-channel measurement (or *trace*) in itself does not typically have the form of a vector of Hamming weights and must pass some preprocessing before being used. We consider this process, called *estimation*, external to the attack itself.

A typical way of implementing the distance function $d(k, z)$ is to perform a power simulation of the cryptographic operation using a hardware model of the cryptographic device assuming the key k, obtain from this simulation a vector of simulated side-channel measurements z^k and return the mean-squared error (or L_2 distance) of the two vectors:

$$d(k, z) = \sum_i \left(z_i^k - z_i\right)^2 \tag{1}$$

We can assume that the measurement z was created from the "optimal" measurement z_c by the addition of some noise vector:

$$z = z_c + e \tag{2}$$

The magnitude of the vector e is defined by the noise model and the performance of the estimator. In this paper we assume a moderately effective estimator and limit our discussions to cases in which the maximum amplitude of e is ± 1 bits in each measurement. An estimator is a *hard estimator* if its outputs are discrete symbols without any confidence information. Under our assumptions the hard estimator will always have a measurement error of either -1, 0 or 1 bits. We can now quantify the errors by considering only P_{err}, the probability that e is nonzero in a given location.

We will now describe several well-known ways of attempting to identify and eliminate noise in decoding problems.

Random Subset Decoding If P_{err} is very low, we can try to sample a random subset of measurement locations. If by chance none of the measurements are errored, we can attempt to recover k_c from the sample and verify its correctness using trial decryption. Assuming a vector with an i.i.d. probability of hard error of P_{err}, the probability that a set of m indices will contain no errors is $(1 - P_{err})^m$. If we assume, for example, that $P_{err} = 0.01$ and that 128 indices are required for an attack to succeed, the overall probability of success is only 27%. For higher error probabilities this method quickly becomes impractical.

Standard Algebraic Attack with Duplication The algebraic attack presented in [18] requires that the measured side-channel information contains no errors. In such a model, a variant of the random subset method can be used: instead of selecting a subset of the data, we can enumerate over all possible locations of errors in the measured data, then create many duplicate instances, each "fixing" the anticipated measurement errors in a certain location and then attempting to carry out an algebraic attack. All duplicate instances are then combined, while we specify to the solver that a single one of the instances needs to be satisfied. Let us assume for example that we have 128 side-channel measurements and we assume that at most 2 locations out of the 128 contain single-bit errors. In this case we can create $\binom{128}{2}$ duplicate instances, each assuming the errors occurred at a certain pair of locations and "fixing" them. We then specify to the solver that only one out of all of the duplicate instances needs to be satisfied. While most of these duplicates will be unsatisfiable (or result in a wrong recovered key), in one of them the measurement error will indeed be cancelled out by our guess, leading to a successful key recovery. The duplication method is obviously only suitable for a very small amount of errors, since the number of additional instances grows exponentially with the amount of anticipated errors.

Iterative Methods If the cipher uses the key bits sequentially (bit by bit) in the encryption or decryption process, an iterative Viterbi-like algorithm [20],

which is described in detail in [11], can be used to recover errors. The Viterbi algorithm's main parameter is its data structure size, which controls the number of key candidates the algorithm maintains during its operation. Letting this size approach $2^{keysize}$, we can treat the iterative algorithm's output as an effective ordering of all key candidates with increasing distance from the measured side-channel information z. The index of the correct key candidate in this ordered list can be an indication of the effectiveness of the iterative approach for solving this specific problem.

The main disadvantage of the iterative method is that it operates in a greedy manner, and cannot return to a key candidate once it has been disqualified. Essentially, this limits the amount of usable side-channel data to a single use of each key bit. In addition, *diffusion* elements (such as the AES MixColumns operation) highly complicate the operation of iterative methods, since many state bits change almost simultaneously and affect every side-channel measurement.

3 Handling Errors by Pseudo-Boolean Representation

3.1 Side-Channel Analysis as a Pseudo-Boolean Problem

Before we present our approach, let us return to the fundamental problem of side-channel analysis, which can be described as follows:

> Given the algorithmic description of a cryptographic algorithm, the physical power model of the device under attack and the side-channel measurements, output a key assignment for which the expected side-channel information is as close as possible to the measured side-channel information.

When written in the above form, it is clear to see that side-channel analysis is naturally represented as an **optimization problem:**

> Find the **minimal** assignment to an **error vector** such that it is possible for the **cryptographic algorithm**, operating under a certain unknown **key** and in a certain **physical power model,** to produce the **measured side-channel information** affected by this error**.**

We call the class of attacks which performs cryptanalysis using an optimizer instead of a solver **Tolerant Algebraic Side-Channel Analysis (TASCA)**.

3.2 An Introduction to Pseudo-Boolean Optimizers

The field of pseudo-Boolean optimization (PBOPT) problems is a special case of integer programming problems [5]. Stated informally, a PBOPT instance consists of an *objective (goal) function* and a series of *inequality constraints*, both of which are defined over some set of Boolean variables. A solution to the PBOPT instance must satisfy all inequality constraints while minimizing the objective

function. Unlike standard Boolean satisfiability (SAT) problems, a PBOPT problem instance admits multiple solutions, choosing the one solution that minimizes the objective function.

As stated formally in [3], a linear PB problem is an optimization problem over n binary (Boolean) variables $x_1 \cdots x_n$ having the following form:

$$\min c^{\mathrm{T}} x \tag{3}$$

$$Ax \geq b \tag{4}$$

$$x \in \{0,1\}^n \tag{5}$$

where all the coefficients are signed integers: $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^n$. The term $c^{\mathrm{T}} x$ is the *objective function* and the row inequalities in $Ax \geq b$ are the *linear constraints*. The solvers we are interested in can also accept *non-linear constraints* of the form $\sum_{i=1}^{t} d_i \prod_{j=1}^{k} \ell_{i,j} \geq r_i$, where $\ell_{i,j} \in \{x_{i,j}, \bar{x}_{i,j}\}$. Because all coefficients are signed values, equality constraints (of the form $\sum d_i \prod \ell_{i,j} = r_i$) and less-than constraints (of the form $\sum d_i \prod \ell_{i,j} \leq r_i$) can also be reduced to the above form.

Because of their relation to both SAT, linear programming, and integer programming, PB instances can be solved using a variety of approaches. Some solvers attempt to compile the PB instance into a SAT instance and apply a standard SAT solver, possibly multiple times; others map the problem into an integer programming instance; some solvers use a hybrid approach, combining the best features of the two.

The pseudo-Boolean description language is very expressive and allows relatively complex constraints to be described quite efficiently. Notably, each errored side-channel measurement can be efficiently written down as a single equation.

The solver we chose to use is SCIPspx version 1.2.0 [3,4,2]. SCIPspx won the first prize for non-linear optimizer in the Pseudo-Boolean Evaluation Contest of SAT 2009 [15]. SCIPspx solves the optimization problem by using integer programming and constraint programming methods. It performs a branch-and-bound algorithm to decompose the problem into sub-problems, solving a linear relaxation on each sub-problem and finally combining the results. The linear relaxation component of SCIPspx is the standalone LP solver SoPlex [21].

3.3 Elements of a TASCA Equation Set

To represent a side-channel attack as a PB optimization instance a TASCA equation set is written, consisting of the following four sections:

1. **A general description of the cryptographic algorithm as a set of equations**: The cryptosystem is described by writing down internal state transformations leading from plaintext to ciphertext. The specification is very hardware-minded, with each state bit/memory element (flip-flop) typically represented as a sequence of variables representing its evolution in time, and each combinational element (gate) finding its way into an equation connecting the variables. For example, the AES state has 16 bytes, each of which

changes its value 4 times in each round (other than the first and the last). This means that the state of each subround is represented by 16×8 binary (0-1) variables, for a total of $16 \times 8 \times 41 = 5248$ variables for an entire AES encryption. There will also be variables for every key bit and every subkey bit, and a set of equations representing the subkey expansion.

2. **An assignment of any known inputs to the algorithm:** These can be known plaintext or ciphertext, or even more subtle hints such as the relationship between two consecutive unknown plaintexts.
3. **A specification of the measurement setup:** The actual side-channel measurement is mapped to the internal state according to the structure of the physical hardware device. For example, an 8-bit microcontroller-based implementation will typically leak the Hamming weight of individual state bytes as they are accessed, while a parallelized ASIC will typically leak the Hamming distance between the former and present values of all bits in the device's internal state. Note that both in the case of Hamming distance and in the case of Hamming weight the measurement equation consists of an equality between a sum of Boolean variables on one side and an integer value on the other – $\sum_j state_{i,j} = m_i$, where $state_{i,j}$ is the value of state variable j at time i and m_i is the side-channel measurement at time i. This form of equation is natural to write down using the PB syntax. It should be noted that when attacking the same cipher running on different target architectures, the measurement setup is usually the only section of the equation set which needs to be modified.
4. **A set of potentially errored measurements:** This section matches the measurements described in the previous section to actual outputs of the estimation phase. As stated previously, the main point of the TASCA approach is to allow errors in the estimation. This is done in practice by adding additional *error variables* to the above-mentioned measurement equations. These error variables are used to cancel out errors in the measurements. In our implementation we included two error variables per measurement (one with a plus sign and one with a minus), which allow the true side-channel value to be within ± 1 bits of the measured one. It should be noted that this section is the only part of the equation set which tolerates errors (all other sections are explicitly defined), and that this section only accounts for 1% to 5% of the entire set of equations for the cryptosystems we tested.

In addition to the equation set, the solver is provided with a **objective function** which it is required to minimize. In our case, our objective is to use as few error variables as possible.

4 An Attack on Keeloq

Keeloq is a block cipher which is most commonly used in remote keyless entry (RKE) systems, e.g. for cars. We chose to attack this cipher first since it has a very simple round structure which is relatively easy to represent as equations. Furthermore, a reduced version of Keeloq (using 140 rounds instead of the

full 528) was already broken using standard algebraic techniques in [7] without requiring side-channel inputs. In [10] a physical ASIC implementation of the Keeloq cipher was shown vulnerable to a standard DPA attack, an attack we were also able to reproduce.

Because it only operates on a single bit of the key in each round, Keeloq is very effectively attacked using the iterative approach described in 2.2.

4.1 The Keeloq Algorithm

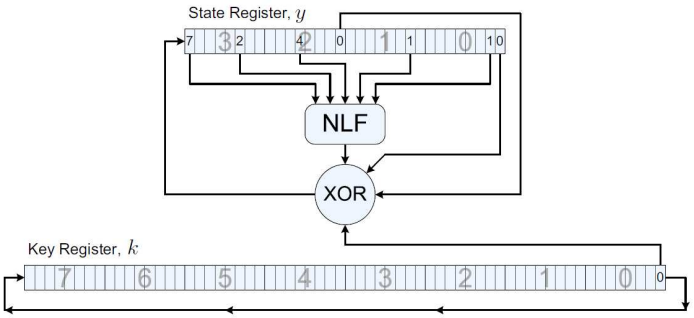

Fig. 1. Structure of the Keeloq cipher (taken from [10])

The Keeloq algorithm [9] is a block cipher designed for efficient hardware implementation. Keeloq has a block size of 32 bits and a key size of 64 bits. As shown in Figure 1 (taken from [10]), its main components include an internal state register (32 bits) and a non-linear feedback function (NLF). In each round of the cipher, NLF operates on five bits from the cipher's current state. The output from NLF is mixed with some prior state bits and with one of the 64 key bits and finally shifted back into the state register. To perform encryption, the plaintext is loaded into the state register, the key is loaded into the key register, and the entire system is clocked for 528 rounds. After these 528 rounds the state register contains the ciphertext. To perform decryption the ciphertext is placed in the state register and the system is clocked 528 times in the opposite direction. The progression of the state register is typically modeled as a vector of bits $S_0 \cdots S_{559}$, with $S_0 \cdots S_{31}$ being the plaintext and $S_{528} \cdots S_{559}$ the ciphertext.

4.2 An Equation Set for Keeloq

As stated in Section 3.3, a TASCA equation set consists of four elements - the algorithm, the inputs, the measurement setup and finally the (potentially errored) measurements.

The algorithmic description of a single Keeloq round is a simple set of 2 PB equations:

$$NLF_i = NLF(S_{i+31}, S_{i+26}, S_{i+20}, S_{i+9}, S_{i+1}) \tag{6}$$

$$S_{i+32} = NLF_i \oplus S_i \oplus S_{i+16} \oplus K_{i \bmod 64} \tag{7}$$

The function NLF is a 5-to-1 non-linear function defined such that $NLF(a, b, c, d, e)$ is bit number $abcde_b$ (binary) of the hexadecimal constant $3A5C742E$, where bit 0 is the least significant bit. It has no efficient linear or algebraic representation, and is represented as a single disjunctive normal form (DNF) equation based on the function's truth table (see the extended version of this paper for more details). The XOR function on 4 variables (effectively 5, since we realize the function $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 = 0$) is also represented by a single equation. Each of these equations is repeated 528 times to lead from the plaintext $(S_0 \cdots S_{31})$ to the ciphertext $(S_{528} \cdots S_{559})$.

In its most common mode of operation, Keeloq uses *rolling codes* which mandates that the ciphertext is known to the attacker but not the plaintext. Accordingly, the only known input to our solver was the ciphertext.

If we assume Keeloq is implemented on an ASIC, the power traces tend to be correlated with the *Hamming distance* of the entire 32 bits of the state register between the current round and the previous round (a similar attack can also be mounted if the device leaks the Hamming weight). To put this into equation form, we define the Hamming distance between each two consecutive bits of the state progression and group them in sets of 32. Finally, we add two additional Boolean variables to the measurement sums to allow for errors:

$$hd_i = S_i \oplus S_{i-1} \tag{8}$$

$$HD_i = \sum_{j=i}^{i+32} hd_j \tag{9}$$

$$\widehat{HD_i} = HD_i + e_i^+ - e_i^-$$

The number of rounds for which we produce side-channel measurement equations (m_{sc}) is a configuration parameter of the system: the following subsection shows how to select a proper value for m_{sc}. A Keeloq key recovery instance with side-channel measurements equations applied to the final $m_{sc} = 90$ rounds of encryption contains a total of 428 equations and has a file size of about 140K. A partial listing of a sample PB instance is provided for reference in the extended version of this paper.

4.3 Attack Results

We performed a power simulation of 300 ASIC-based Keeloq decryptions, corrupted the simulated power measurements with different probabilities of error and submitted them to the SCIP PB-solver. For each decryption and tested error

probability, we selected $P_{err} \times 528$ rounds and corrupted the side-channel values measured in those rounds (specifically, the device-total Hamming distance) by ± 1. The attack used the 64-bit version of SCIPspx 1.2.0 on a quad-core Intel Core i7 950 at 3.06GHz with 8MB cache, running Windows 7 64-bit Edition. In each experiment the solver was asked to recover the 64-bit key from the errored side-channel outputs produced by the final 90 rounds of encryption.

Figure 2 shows our results. For reference we also show the performance of an iterative attack on the final 64 rounds of the encryption using comparable bit error rates. We emphasize that the attack is on the **full** 528-round cipher, even though it uses only a subset of the measured side-channel data.

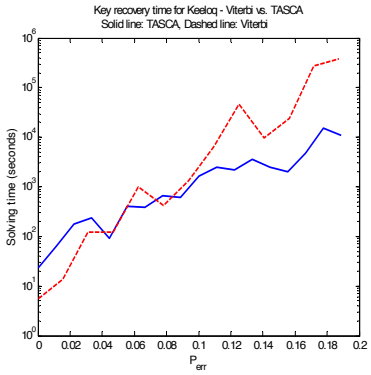

Fig. 2. TASCA key recovery from the final 90 rounds of Keeloq

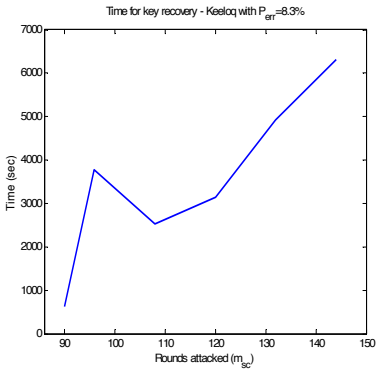

Fig. 3. TASCA speed as a function of m_{sc}

It can be seen that the solver was able to find the key even with $P_{err} = 18.8\%$ with an average running time of 3.8 hours per instance, and that the time grows super-linearly with the error probability. The TASCA solver is 10 to 100 times faster than the iterative solver on instances with $P_{err} \geq 11\%$.

We also noted it was important to properly choose the number of side-channel measurements passed to the solver (m_{sc}). When too few rounds were passed, the optimal solution found was not necessarily the correct key. When too many rounds were passed, the computational burden involved slowed down the solver, as shown in Figure 3. In the case of Keeloq a good tradeoff was $m_{sc} = 90$ rounds which provided enough information to find the key in nearly all of the cases. For lower m_{sc}values the solver returned incorrect results for at least 25% of the instances.

As an aside note, the iterative decoder, which struggled with single-trace key recovery, had much better performance when attacking multiple traces. The algebraic solver did not perform as well with additional inputs, since each additional trace significantly increased the size of the equation set.

Table 1. Instance size and performance of straight encryption

	Keeloq	AES (LUT)	AES (Canright)
Instance file size	553K	32873K	12569K
# of equations	1153	27344	93090
# of variables	13825	171208	229008
# of constraints	13825	173640	231506
Encryption time (sec.)	2.59	61.07	245.45

5 Preliminary Results on AES

5.1 The AES Algorithm

We chose to model our device under attack as naïve 8-bit microcontroller implemention of AES-128[8], is a block cipher with a 128-bit key and 16-byte (or 128-bit) input blocks. To perform encryption, the plaintext is first fed into a 16-byte state register. This state register is then manipulated 41 times during the sequence of the 10 rounds of AES-128 to produce the ciphertext. There are 4 types of manipulations: SubBytes, AddRoundKey, ShiftRows and MixColumns, with SubBytes being the only non-linear operation in AES. AddRoundKey is performed 11 times during encryption, first with the supplied secret key and then 10 times with round keys derived from the secret key using a non-linear process which uses the SubBytes process as well.

5.2 An Equation Set for AES

The AES hardware realization can be modified and optimized in a variety of ways. Specifically when dealing with the S-box component of AES, which performs the SubBytes operation, there are a variety of hardware implementations offering various tradeoffs between better speed and more efficient hardware consumption.

Our first TASCA representation of AES implementation was based on a port of an OpenCores VHDL AES code [19]. This implementation models the S-box as a lookup table (LUT), leaving the compiler with the task of optimizing it to a minimal hardware footprint. A second TASCA representation was based on the efficient composite field representation of the S-box designed by Canright [6]. In this design, the S-box input is manipulated under a more efficient basis representation.

Table 1 summarizes the performance of the two cipher implementations, with the performance of the Keeloq encryption provided as reference. Since the encryption was described as an equation set, performance was similar whether the plaintext, the ciphertext or any intermediate state was supplied. Similarly, any round key can be substituted for the secret key with no effect on performance. Analyzing the results, it appears that the performance of the solver is dominated by the number of equations under consideration and not by the complexity of the equations themselves. This may be a property of the SCIP solver, and not

Table 2. A TASCA attack on the AES key expansion phase

Rounds	AES (LUT)			AES (Canright)		
	instance size	# of equations	time (sec)	instance size	# of equations	time (sec)
1	765K	164	11	208K	1484	193
2	1529K	308	1341	414K	2948	10800
3	2293K	452	1690	620K	4412	345600

of PB optimizers in general, since SCIP essentially performs a search over the tree of equations. Specifically, other PB solvers which compile their inputs into SAT instances may show better performance using the Canright S-box, since its reduced hardware complexity should make it easier to simulate.

Surprisingly, the solver had a very hard time inverting the SubBytes and MixColumns operations given their algebraic description. We found out that including equations for both SubBytes and for inverse SubBytes (that is, one equation stating that $S_{i+1} = SubBytes\,(S_i)$ and another stating that $S_i = SubBytes^{-1}\,(S_{i+1})$) sped up the solver dramatically.

5.3 Initial Results

To date, the only attacks we have run on AES are reconstructions of the SPA attack of Mangard on the key expansion algorithm, as described in [13], without any errors. A secret key was recovered from 1, 2 and 3 rounds of expansion (16, 32 and 48 Hamming weights[3]). The key expansion was modeled using both S-box representations. The results are summarized in Table 2.

The time performance of the solver was worse than we estimated. We were also surprised to find that Canright representation yielded longer running times than the LUT representation, despite having instances that are 3 times smaller. Understanding these phenomena is a topic of future work.

6 Open Issues

6.1 Full Attack Against AES and Other Ciphers

Our preliminary results thus far show that single-trace side-channel attacks against AES can be represented as PBOPT problems, and that the representations vary dramatically in size and complexity depending on the hardware implementation we start with. Furthermore, TASCA running time was not correlated with instance size - in fact the more compact Canright representation produced run times an order of magnitude slower than those produced by the LUT representation. We plan to try and better understand which instance types lead to faster TASCA attacks.

[3] The published results in [13] show that 40 recovered Hamming weights are enough to uniquely determine the secret key

6.2 Better PB Solvers

The authors do not claim to be experts in the design and usage of PB solvers. In fact, the SCIP tool which we used has hundreds of configuration options which were left at their default values. It appears that the performance of the solvers can be increased by quite a large factor using careful design – the fact that a simple AES encryption took 60 seconds on our unoptimized platform is especially surprising. Since these solvers rely on heuristics to improve their performance, a set of heuristics for cryptanalysis needs to be developed. With a proper choice of heuristics we hope the performance of these attacks can be increased by several orders of magnitude, either by using SCIP or by evaluating a different PB solver. To this end, we have shared our cryptanalytic instances with the PB design community [15].

6.3 TASCA as Part of the Design Tool Chain

The specification language used to define PB optimization problems is rich enough to allow description of arbitrary Boolean circuits. It seems possible to write a compiler that receives a hardware description in a high-level language such as VHDL [1] and outputs a PB-solver instance. Such a tool can be made part of a secure hardware design workflow, allowing designers to evaluate the susceptibility of their designs to side-channel attacks. By performing TASCA attacks with different subsets of the side-channel information, designers can assess the risk caused by exposure of various components of the internal state and so decide which components need a higher level of protection.

7 Conclusion

We showed a new attack methodology called *Tolerant Algebraic Side-Channel Analysis (TASCA)*. Our methodology transforms a single-trace side-channel analysis problem into a *pseudo-Boolean optimization problem* (PBOPT) form. The PBOPT syntax allows a very natural representation of measurement errors. We showed that using our approach we are able to mount successful single-trace attacks against real ciphers, even in the presence of realistic error rates.

Acknowledgements. Parts of the research described in this paper have been supported by the Austrian Science Fund (FWF) under grant number P22241-N23 ("Investigation of Implementation Attacks"). The authors wish to thank the anonymous reviewers for their encouraging and insightful comments.

References

1. IEEE standard VHDL language reference manual. *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pages c1 –626, 26 2009.

2. T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.
3. T. Berthold, S. Heinz, and M. E. Pfetsch. Nonlinear pseudo-boolean optimization: Relaxation or propagation? In O. Kullmann, editor, *SAT 2009*, volume 5584 of *LNCS*, pages 441–446. Springer, 2009.
4. T. Berthold, S. Heinz, M. E. Pfetsch, and M. Winkler. SCIP – solving constraint integer programs. SAT 2009 competitive events booklet, 2009.
5. D. Bertsimas and R. Weismantel. *Optimization Over Integers*. Dynamic Ideas, 2005.
6. D. Canright. A very compact S-Box for AES. In J. R. Rao and B. Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 441–455. Springer, 2005.
7. N. Courtois, G. V. Bard, and D. Wagner. Algebraic and slide attacks on KeeLoq. In K. Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 97–115. Springer, 2008.
8. J. Daemen and V. Rijmen. AES proposal: Rijndael, 1998.
9. S. Dawson. Code hopping decoder using a PIC16C56. Microchip confidential, leaked online in 2002, 1998.
10. T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. Manzuri Shalmani. On the power of power analysis in the real world: A complete break of the Keeloq code hopping scheme. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 203–220. Springer, 2008.
11. C. Karlof and D. Wagner. Hidden Markov model cryptoanalysis. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 17–34. Springer, 2003.
12. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.
13. S. Mangard. A simple power-analysis (SPA) attack on implementations of the AES key expansion. In P. J. Lee and C. H. Lim, editors, *ICISC 2002*, volume 2587 of *LNCS*, pages 343–358. Springer, 2002.
14. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
15. V. Manquinho and O. Roussel. Pseudo-boolean competition 2009. Online, July 2009.
16. F. Massacci and L. Marraro. Logical cryptanalysis as a SAT problem. *J. Autom. Reason.*, 24(1-2):165–203, 2000.
17. N.R. Potlapally, A. Raghunathan, S. Ravi, N.K. Jha, and R.B. Lee. Aiding side-channel attacks on cryptographic software with satisfiability-based analysis. *IEEE Trans. on VLSI Systems*, 15(4):465 –470, april 2007.
18. M. Renauld, F.-X. Standaert, and N. Veyrat-Charvillon. Algebraic side-channel attacks on the AES: Why time also matters in DPA. In C. Clavier and K. Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 97–111. Springer, 2009.
19. H. Satyanarayana. AES128 package. Online, December 2004.
20. A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260 – 269, Apr 1967.
21. R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996.