Investigating Profiled Side-Channel Attacks Against the DES Key Schedule

Johann Heyszl[1][0000−0002−8425−3114], Katja Miller[1], Florian Unterstein[1][0000−0002−8384−2021], Marc Schink[1], Alexander Wagner[1], Horst Gieser[2], Sven Freud[3], Tobias Damm[3][0000−0003−3221−7207], Dominik Klein[3][0000−0001−8174−7445] and Dennis Kügler[3]

[1] Fraunhofer Institute for Applied and Integrated Security (AISEC), Germany,
firstname.lastname@aisec.fraunhofer.de

[2] Fraunhofer Research Institution for Microsystems and Solid State Technologies (EMFT), Germany, horst.gieser@emft.fraunhofer.de

[3] Bundesamt für Sicherheit in der Informationstechnik (BSI), Germany,
firstname.lastname@bsi.bund.de

Abstract. Recent publications describe profiled single trace side-channel attacks (SCAs) against the DES key-schedule of a "commercially available security controller". They report a significant reduction of the average remaining entropy of cryptographic keys after the attack, with surprisingly large, key-dependent variations of attack results, and individual cases with remaining key entropies as low as a few bits. Unfortunately, they leave important questions unanswered: Are the reported wide distributions of results plausible - can this be explained? Are the results device-specific or more generally applicable to other devices? What is the actual impact on the security of 3-key triple DES? We systematically answer those and several other questions by analyzing two commercial security controllers and a general purpose microcontroller. We observe a significant overall reduction and, importantly, also observe a large key-dependent variation in single DES key security levels, i.e. 49.4 bit mean and 0.9 % of keys < 40 bit (first investigated security controller; other results similar). We also observe a small fraction of keys with exceptionally low security levels that can be called *weak* keys. It is unclear, whether a device's side-channel security should be assessed based on such rare *weak* key outliers. We generalize results to other leakage models by attacking the hardware DES accelerator of a general purpose microcontroller exhibiting a different leakage model. A highly simplified leakage simulation also confirms the wide distribution and shows that security levels are predictable to some extend. Through extensive investigations we find that the actual *weakness* of keys mainly stems from the specific switching noise they cause. Based on our investigations we expect that widely distributed results and weak outliers should be expected for all profiled attacks against (insufficiently protected) key-schedules, regardless of the algorithm and specific implementation. Finally, we describe a sound approach to estimate actual 3-key triple-DES security levels from empirical single DES results and find that the impact on the security of 3-key triple-DES is limited, i.e. 96.1 bit mean and 0.24 % of key-triples < 80 bit for the same security controller.

Keywords: DES · 3-DES · SCA · side-channel attack · key schedule · weak keys

1 Introduction

Most side-channel attacks (SCAs) target the main data-path of cryptographic engines, where input data *and* round keys are processed. Differential attacks can generally be considered as most powerful because they use side-channel observations with different input data to recover keys using statistical tools. The matter becomes significantly more difficult for attackers when implementation parts should be targeted where no such differential analysis is possible. The transfer of constant secret keys over internal buses and its handling during the key schedule are examples for this. Attacks in such cases are usually profiled and performed by comparing previously acquired templates to new side-channel observations. They usually require a higher measurement precision. This can e.g. be achieved using high-precision EM measurements [HMH+12].

Wagner et al. [WH17, WHG17, WH18] published results of mounting a profiled side-channel attack against the key schedule of a "DES hardware accelerator" in a "security controller" product. Their SCA attack provides partial information about the key and the key must be recovered through a subsequent brute-force search. The effort required by the brute-force search is represented by the remaining security level or guessing entropy of a key in bit (cf. definition in Section 5.4). The authors find a significant reduction of security levels, but it appears to be highly dependent on the key. This leads to a distribution of results and implies the existence of particularly *weak* keys. Their results raise interesting questions, both for the specific case of their device, and in general. Most of these questions are left unanswered in the original series of papers. Several of their choices during analysis seem heuristic or ad-hoc, calling for a re-assessment through a second investigation.

In this work, we independently perform an investigation on a comparable device and provide carefully reasoned and fully documented results including new insights and answers to the following important questions:

1. *Can the results from Wagner et al. on single DES be reproduced independently and what are the actual results for a commercial security controller? Are there really weak keys?*

   We perform an analysis of a comparable device and carefully argue every choice of algorithm. We systematically attest that the device exclusively leaks the XOR difference between key bits during the DES key schedule, which allows us to make reasonable algorithm choices. The results confirm several statements of Wagner et al. Specifically, a reduction of security levels to a similar extent is achieved for single DES, i.e. the average remaining security level is reduced to 48.2 bit (from 56 bit) after a comparable amount of traces per key. Also, some keys are significantly *weaker* than others with 1.8 % of keys < 40 bit (when using 3 traces per key). In terms of worst case, tested with key bits being either all ones or all zeros (note that those keys are also cryptographically weak and thus an unrealistic choice), we found security levels as low as only 2 bit for those two particular keys.

   Contrarily, the resulting security levels of DPA attacks against AES implementations is largely independent of the key's value.

2. *If weaker individual keys exist for single DES, what is the impact on 3-key triple-DES implementations?*

   Three-key triple-DES is the last remaining 'acceptable' use of the DES algorithm. Note, however, that the German BSI does generally *not recommend* using DES [Bun19]. The impact on three-key triple-DES was left unanswered by Wagner et al. We describe a sound approach to derive security levels for 3-key triple-DES based on single DES security levels. This is done by generalizing a meet-in-the-middle attack for side-channel results which is not straight-forward. Single trace attacks against single DES in one of the analyzed devices resulted in a distribution of security levels with 49.4 bit mean and 0.9 % of keys < 40 bit. The derived distribution of security levels for 3-key triple-DES after the side-channel attack (112 bit cryptanalytic security) has a mean of 96.1 bit while only 0.24 % of key-triples lead to a security level < 80 bit. Although this reduction of security levels is significant on average, the number of key-triples < 80 bit is still comparably low. If an attacker uses ≈ 900 measurements per key (no improvement after ≈ 400 traces), the percentage of key-triples < 80 bit increases to 6.3 %. However, a security level of < 70 bits is still rare and achieved in only 0.32 % of cases. This means that an attacker faced with actual devices will only reach a brute-force effort < 70 bit for every 300-th device, given he performs a > 400-trace attack on every device.

3. *Are there SCA-weak DES keys? What is the reason and is it device-specific?*

   Our investigations confirm the widely distributed security levels and dependence on the key value, including significantly *weak keys*[1]. An important question to ask is whether the widely distributed security levels and occurrence of weak keys are due to properties of the device, noise, or of algorithmic origin. How are they influenced by the key-schedule algorithm and exhibited leakage model? This was left unanswered by Wagner et al., as they are e.g. missing repeated measurements of the same keys (for more than one example) to isolate the influence of electrical noise. Through investigations of different devices, i.e. two security controllers and a general purpose controller, including many measurements per key to investigate the noise influence, and a simplified simulation, we show that the *reason for different security levels and weak keys* is in fact not specific to a single type of device, but the *combination of the DES key-schedule algorithm and either one of two frequently encountered leakage models.* The investigated devices, two commercial smart card controllers and a general purpose microcontroller, exhibit completely different leakage models, but show similar results, as does the simulation. Hence, in case of the DES algorithm, an SCA on the key-schedule leads to *weak keys* under different implementation circumstances.

4. *Can the weakness of individual keys after SCA be predicted through simulation?*

   We use a simplified model of the DES key schedule with perfect and equally weighted Hamming distance leakage (XOR leakage) of round keys to perform simulated attacks. This is extremely fast compared to performing actual measurements and significantly more general in the sense that apart from the simplified leakage model, no other properties of the device are used. Interestingly, this simulation reproduces widely distributed security levels and weak keys. The simulation even allows for a rough prediction of security levels for individual keys for the actual device, which also emphasizes the generality of the issue. However, the simplifications seem to prevent precise predictions for a specific device. Additional device-specific parameters to weight XOR transitions could be derived from actual measurements and would likely improve results. This could, in theory, allow to dismiss weak keys before using them based on simulation to improve the worst-case security bound after side-channel analysis. On the other hand this would hypothetically allow to deliberately generate and distribute weak keys for malicious purposes.

   Interestingly, for keys with significantly uneven distributed ones/zeros, the simplified simulation provides a very accurate prediction of measurement results. Keys with low security levels according to this simplified model will likely be weak on all hardware implementations exhibiting XOR leakage. This emphasizes the prediction capability of the model in general. Such particular keys with a Hamming weight close to either the minimum or maximum are, however, statistically rare in case of random number generators for cryptographic key generation, which are usually required to produce

[1] In theoretical cryptanalysis, cipher-specific *weak keys* lead to undesirable behavior, e.g. that encryption and decryption are equivalent (i.e. $enc_k(enc_k(x)) = x$) because all subkeys are equal (and the cipher is a Feistel cipher). The term *weak key* in the context of this work describes a key with a low security level after a side-channel attack.

uniformly distributed ones and zeros (e.g. through post-processing using a block cipher).

5. *How likely is it that other implementations or devices are affected by weak keys?*

   Our investigations show that different leakage models together with the DES key schedule properties lead to widely distributed results and weak keys. Hence, *every device which shows any subkey-dependent leakage (of at least the two confirmed forms, i.e. value and XOR leakage) is likely affected.*

Section 2 recaps on the works of Wagner et al., discusses important aspects from their contributions and differentiates to our work. We systematically explain all crucial choices when attacking key schedules using profiled template attacks in Section 3. The following Section 4 provides necessary information about the DES key schedule and answers the remaining questions regarding template attacks given XOR leakage specifically. Section 5 provides the results and insights of an extensive empirical study of a security controller product and includes the simulation results. Section 6 explains how to derive 3-key triple DES results from single DES evaluations. Section 7 contains the results of an empirical study of a general purpose microcontroller which allows further generalization. Section 8 presents the results achieved after analyzing a second security controller. To aid reproducibility and interpretation of our results, all algorithms and tools used in this work are described in Appendix A.

2 Related work

Wagner and multiple co-authors focused on a "commercially available smartcard" including a DES co-processor in the following contributions published on the IACR's eprint server. To the best of our knowledge, none of the following contributions have been published in a peer-reviewed conference or journal. We discuss their publications in the order of publication date.

2.1 Hu et al., *Ciphertext and Plaintext Leakage Reveals the Entire TDES Key*, 2016

In 2016, Hu, Zhang, Zheng and Wagner [HZZW16] investigate the DES co-processor of a commercially available smartcard running unidentified software. They use an EM measurement setup without stating details on the probe, sampling rate, measurement position, or alignment. Four DES executions are observed within a short frame instead of a single one as expected. As a reason they assume a forward-backward-forward-backward computing DFA countermeasure. They report statistically significant correlations between measurements and Hamming distances between consecutive 32 bit words of the key, plaintext, and ciphertext within few cycles before/after an identified DES operations. They assume an internal 32-bit bus as cause of this leakage. Based on these findings, Hu et al. describe that guessed keys could be tested by using a known plaintext and performing ciphertext correlations for candidates. This would help to verify triple-DES part-keys in case their respective key space is already reduced significantly, an important prerequisite which is addressed in later contributions. However, for the testing of every candidate, a correlation-based side-channel attack would have to be performed. This is only realistic for very short lists of candidates. In summary, it seems difficult to construct an actual attack based on their findings.

2.2 Wagner et al., *Comparative Study of Various Approximations to the Covariance Matrix in Template Attacks*, 2016

In 2016, Wagner, Hu, Zhang and Zheng [WHZZ16] describe further results from a side-channel analysis of the same DES implementation. In this contribution, they perform multivariate profiled template attacks to exploit the distance-based leakage of ciphertext, plaintext and key bytes, which they uncovered in their previous publication [HZZW16]. They find that the XOR distances between key parts can be recovered with a low number of traces (e.g. range of 500). Based on this, they describe a possible attack similar to their previously published one and estimate that (only) 6 to 10 bits per DES key could be recovered. Additionally, they evaluate different statistical models and parameter estimation techniques by choosing different types of covariance matrices and methods to estimate the coefficients. They find that some strategies lead to better results than others. However, in their later attacks on the DES key schedule [WH17, WHG17] they report best results with averaged covariance matrices, which is state of the art and usually referred to as *pooled* covariance matrices.

In this work, a Langer EM probe with unknown specifications is placed 'on top of a DES hard-macro', identified in an unknown manner, and a sampling rate of 5 GS/s is used. Pattern matching is used to extract the four DES operations, and an elastic alignment filter helps to cope with the internal and unsynchronized clock. They record 7 million traces, and use 5 million for profiling after dismissing untypical ones.

2.3 Wagner and Heyse, *Single-Trace Template Attack on the DES Round Keys of a Recent Smart Card*, 2017.

In 2017, Wagner and Heyse [WH17] finally describe a 'single-trace' attack claiming to exploit a newly uncovered internal distance leakage between DES key bits during the DES key schedule of the same commercial smart card. To explain their observation that secret key bits actually leak their XOR distance, they assume that round keys are masked with a value that is constant for each computation. The repetitive handling of bits, which is due to the DES key schedule specification, together with this observation leads to exploitable distance leakage between key bits, which is described in [WH17, Equation (3)]. The authors list all distances between key bits and the rounds they occur in [WH17, Table 5]. Four successive DES operations are exploited in each 'single trace'. As an attack result, they report a remaining entropy of 48.5 bit on average [WH17, Table 6] and few cases with significantly lower remaining entropies. They do not answer the question, whether the large difference in attack outcome is due to key values and the algorithm, or stems from e.g. electric noise.

During the profiling phase, templates are created for subkeys (i.e. their *values*), which are obtained by grouping key bits into subkeys according to occurring transitions in the key schedule. The template size, which is the number of bits profiled together, is 7 bit. (They also test different template sizes; see the subsequent contribution [WHG17].) For each possible bit-value of each subkey, a profile is created using ≈ 4.7 million traces with random keys. Since the actual leakage is a Hamming distance leakage instead of a value-based one, they are forced to choose their subkeys with overlapping bits. See Section 3 for a detailed discussion about this and Section 4.1 explaining why profiling bit-*values* is not reasonable (instead, XORs should be profiled). This overlap approach further creates a problem when combining ranked lists from the template matching of the overlapping subkeys. Their proprietary key enumeration and rank estimation is suboptimal, since they are disregarding probabilities from SCA and ignoring dependencies between overlapping subkeys. Any optimal key rank or enumeration algorithm estimates the combined probabilities for key candidates based on the subkey probabilities [VCGRS12]. The issues are partly addressed in a subsequent contribution [WHG17] where probabilities are used instead of subkey ranks. We discuss established methods in Section 3. The authors also argue how additional sources of leakage, e.g. the Hamming weight of the full key, could be used to improve the attack.

2.4 Wagner et al., *Brute-Force Search Strategies for Single-Trace and Few-Traces Template Attacks on the DES Round Keys of a Recent Smart Card*, 2017

As a follow-up to their previous contribution, Wagner, Heyse and Guillemet [WHG17] describe improved 'brute-force' strategies of enumerating keys based on the results from the template matching described in their previous contribution. The improved strategies are based on lists of ranked candidates. It can be seen as an attempt to key enumeration while still disregarding probability information or likelihoods from the template matching of subkeys, which is unreasonable. In more detail, they derive lists of value-candidates for overlapping subkey parts (e.g. 14 lists of 7 bit templates with 5 bit overlap for one 28 bit register; they call these 'C'-rings) from the template attack. Then as a first step, an *unordered list* of all possible combinations is created while discarding combinations which are impossible due to not-matching values in the overlapping bits (e.g. 2^{27} candidates remain, one bit indistinguishable due to the actual distance model). Subsequently, and disregarding all derived likelihoods or probabilities from the template attack, they sort this list of keys. For this, they use the average, or maximum *ranking* of all subkey parts for every item on the list. From the two registers, a heuristic brute-force mechanism is used to enumerate entire keys.

Results for DES. In a test with $\approx 297k$ single trace attacks (four DES executions per trace) on individual keys, they report an average remaining entropy of 45.65 bit out of 56 bit [WHG17, Fig. 12]. While this is a significant reduction, it is not critical on average. The results for different keys show a distribution with a large variance, however. The $1 \times \sigma$ interval roughly extends down to 40 bits. They try different template sizes and report best results with 9-bit templates (with 7 bit overlap of adjacent templates). All choices lead to similar results. Note that through the overlap, the attack essentially extracts the same information from the trace repeatedly at an increased computational cost.

In [WHG17, Section 3], they attempt to use 'additional' leakage by first adding up trace parts corresponding to the 16 rounds from the four DES executions. Then, the sum of the Hamming distances of all bits of a round is used for a profiled template attack. This requires ≈ 600 templates to model all possible summed Hamming distances which may obviously only convey limited information. Instead of estimating the probability for groups of bits and derive the joint probabilities through a rank estimation algorithm, their strategy is to heuristically modify their 'brute-force search algorithm' by removing all candidates which do not match with the observed Hamming distance with a deliberately chosen error-margin of ± 7 bits. Authors report an average improvement of ≈ 2.5 bit.

Impact on 2-key triple-DES. Wagner et al. estimate 2-key triple-DES results based on their single DES attack results. They take single DES results, and for all possible combinations of individual results, they *square*[2] the higher remaining entropy to derive an estimated 2-key triple-DES result. Doing this, they implicitly dismiss a possible meet-in-the-middle improvement. See Section 6 in this work about how to derive distributions for multi-key encryption from single-key measurements and how to benefit from the meet-in-the-middle approach using side-channel attack results on the example of 3-key triple-DES. They report a distribution with a mean remaining entropy of 96.47 bits [WHG17, Fig. 17]. The occurrence of low entropy 2-key combinations is extremely rare. They also report

[2] Note that authors state 'double' and most probably mean doubling the bit-representation

2-key triple DES results (for different template sizes)[WHG17, Table 2]. According to their table (template size 9 bit), 0.068 % are < 70 bit, 0.0011 % are < 60 bit, 0.0000088 % are < 50 bit. The decline is exponential. The authors convert this probability of occurrence into an 'effort' measured in bit to achieve a combined attacker effort in bits. However, given each device uses one specific key, attackers would actually need to change devices for this. The required efforts for changing devices are significantly higher than brute-forcing key candidates offline. Hence, a combination of the two complexities seems misleading. They report results indicating a combined security level exceeding 70 bit for all cases, even for cases where the estimated remaining entropies are lower than 50 bit, because the probability of occurrence is low. Based on the fact that one key is used twice in a 2-key triple-DES and using what they call 'total Hamming distance leakage', they claim that results could be improved by several bits. The presented estimates for these improvements are not convincing, as they are partly based on a very low number of observations; e.g. three observations in [WHG17, Table 4].

Bounds for Security Levels through Simulation. Wagner et al. investigate the reason for weak keys using a simulation [WHG17, Section 5]. While their argumentation is disregarding several important aspects (e.g. influence of noise is not investigated thoroughly to rule it out as cause for the distribution), their main claims, i.e. that some keys are weaker than others and that this depends on the key bit values, is in line with our findings.

Wagner et al. attempt to determine the remaining entropy of given keys after a simulated side-channel observation without electrical noise. They assume that the observable leakage is the Hamming distance between successive round-keys and that the attacker is able to exploit this perfectly. Keys k and k' with equal traces, i.e. summed Hamming distances per each round key update time-point, are still not distinguishable for an attacker. By counting the number of keys k' with identical leakage they try to estimate a theoretical lower bound for the security level for every given key k. This is reasonable under the strong attacker assumptions mentioned above. However, note that this also implicitly assumes attackers to be capable of profiling and attacking large parts of the key at once. Counting such collisions is difficult, however.

Each side-channel observation in this simulation is fully characterized by 15 values of Hamming distances (HD) between successive round-keys. It is represented by a function $\text{dist15}(k) = \{\text{HD}(\text{subkey}(i), \text{subkey}(i+1))\}_{i=1}^{15} \in [0, 48]^{15}$, which maps each key to its leakage. Based on the DES specification, Wagner et al. rightly claim that the leakage is the sum of contributions from both equally structured registers, $\text{dist15}(k) = \text{dist15}_C(k) + \text{dist15}_D(k)$ for each key k. In order to make counting of collisions feasible, they employ a two-stage matching with a systematic error. In the first stage, when given a key k, and while going through possible candidates k', they keep all keys k', such that $||\text{dist15}_C(k)||_1 = ||\text{dist15}_C(k')||_1$ and $||\text{dist15}_D(k)||_1 = ||\text{dist15}_D(k')||_1$. Here $||v||_1 = \sum |v_i|$ denotes the L1-norm of a vector v. In other words, they only keep all k' where the simplified L1-norm of the first part (register C) matches the simplified L1-norm of the first part of k, and the same for the second part (register D). This reduces the number of candidates to a manageable set. They then count the true collisions ($\text{dist15}(k) = \text{dist15}(k')$) in the remaining set. Unfortunately, their first stage already discards possible matches introducing the error. Take as an example the two keys $k_1 =$ `0x01E001E0_01F101F1` and $k_2 =$ `0x011F011F_010E010E`. Both lead to identical leakage since the roles of the two registers are simply swapped:

$$\text{dist15}_C(k_1) = \text{dist15}_D(k_2) = [24, 0, 0, 0, 0, 0, 0, 24, 0, 0, 0, 0, 0, 0, 24] \quad \text{and}$$
$$\text{dist15}_D(k_1) = \text{dist15}_C(k_2) = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].$$

However, this key-pair (along with many others) would be missed by the simulation of Wagner et al. because the simplified L1-norm of the registers C and D do not match.

This means that they *systematically underestimate the security levels.* As a conclusion, this means that their given results cannot be used as a reference for the theoretical attack potential and lower bound for the security level of the DES key-schedule.

They performed the above described simulation for a random selection of 8k keys which results in an average remaining entropy of only 15.0 bit [WHG17, Fig. 42]. They used the 10 weakest keys (0.135 % of 8k) from their simulation for an actual attack which leads to an average security level of 41.6 bit. The results are lower than the overall average of 46.16 bit [WHG17, Fig. 11] but from the discussion above, it is already expected that the low security levels from the simulation are faulty. The simulation results do not represent *achievable* lower bounds.

Influence of independent noise on key weakness. Wagner et al. perform 32k repeated attacks for (only) one fixed key [WHG17, Section 6]. They find that even for this fixed key, the results exhibit a near-Gaussian distribution with a large variance. While not noted by the authors, this is interesting because it could indicate that attack results are in fact very dependent on the trace-specific random noise in addition to key properties. Their results also show that the remaining entropy of this single key converges to a certain level when increasing the number of traces used for the attack. An analysis of more keys is missing.

2.5 Wagner and Heyse, *Improved Brute-Force Search Strategies for Single-Trace and Few-Traces Template Attacks on the DES Round Keys*, 2018

As an improvement to their previous work, Wagner and Heyse [WH18] describe modifications to their enumeration of key candidates. Their setting is equivalent to previous papers (measurement setup, 11-bit templates, high overlap, pooled covariance matrices). Authors report that a template size of 11 bit has been used, however, also report that smaller sizes did not lead to worse results. In this work, they only use previously identified 378 'weak keys' (0.135 % of total keys) which led to the best results in their previous work. Wagner et al. only record single measurements per key (note that their single trace attack actually contains four DES executions, however).

For the improved search strategy, they create a list of all possible combinations of value-candidates for subkeys after template matching as done in their previous work. Also, they remove candidates where the overlapping bits do not match. As a novelty, they then sort the remaining candidates according to the product of matching probabilities of key parts (precisely through summing after taking the logarithm) using Quicksort. They do so for both registers (e.g. half keys) separately. They disregard dependencies which are inherent due to the overlap. (Note that both state of the art key enumeration algorithms mentioned in Section A.5 as well as their approach require independence between the key parts for optimal results.) They then use an incrementally increased threshold to repeatedly test combinations of subkeys with probabilities below that threshold. In this search, they weight the probabilities in the two separate lists slightly different to represent alleged differences in leakage between the two registers. In addition, they also incorporate 'total Hamming distance leakage' as described in a previous contribution which accounts for roughly 1 bit of the reduction. As a result, they claim that the average remaining entropy (it remains unclear, however, how this is determined precisely) is lowered from ≈ 41.7 to ≈ 38.8 bits for this set of 378 weakest keys with values ranging from ≈ 16 to ≈ 52 bit. They explicitly claim achievable security levels as low as 2 bit, based on the flawed simulation described in their preceding paper.

2.6 Summary

Many described assumptions, decisions, steps, and algorithms by the authors are either not respecting the state of the art, or argued in a manner which is unclear or not satisfying. This makes it impossible to assess the impact of the attack on the security of DES implementations. Nonetheless, their attack reduces the key entropy and demonstrates that there is exploitable leakage. The generality of the threat to other implementations, devices and cryptographic algorithms is left open by this series of papers.

3 Profiled attacks to recover key bit values - Leakage models, classification, enumeration

The aim of profiled side-channel attacks is the classification of values from (few) noisy observations using statistical methods after a preliminary profiling. Key-dependent intermediate values are often targeted during a cryptographic computation. In the context of attacks on a key schedule, a likely target are key bits directly. A leakage model specifies the recoverable values as variables and their functional dependencies. For the observable variables, the statistical parameters are then estimated during profiling and used for classification. It is important to derive a solid understanding about the leakage model of the attacked secret to determine the best or even optimal algorithms for classification. In most cases only parts of the key (here called subkeys, usually bytes or groups of key bits) can be classified at once due to resource constraints. The result is a list of candidates including probabilities. The leakage model and derived classification approach also influence how the enumeration of candidates for the entire key based on subkey classification results can be performed.

Given the context of an attack against the DES key schedule and considering general implementation choices (see Section 4), a small set of leakage models is reasonable and should be considered. First, bits may leak their value directly. In case of a hardware implementation, a transition between successive values in a register may also be a valid model of the leakage. Best or even optimal classification and enumeration results can only be achieved if the actual leakage model of a device is investigated thoroughly and choices made accordingly. In the following, we discuss relevant leakage models and their implications on the choice of suitable algorithms.

3.1 Different possible leakage models of key bits

The following four subsections describe different possible leakage models of key bit variables k_i which are targeted, e.g. during an attack on the key schedule. Since key bits are generally chosen randomly from a uniform distribution, there is no a priori dependency between them. Therefore, they are depicted as independent variables in the following figures.

3.1.1 Independent value leakage

The simplest form of side-channel leakage is when the value of each bit is leaked directly (i.e. single bit Hamming weight, or bit-weight) and independently, which we refer to as *value leakage*. Figure 1 depicts such a case where every bit leaks its value on a side-channel trace in a way so that by looking at a part of the trace (i.e. a number of time-samples denoted by $\mathcal{L}_j$) the different bits can be observed independent from each other.

If value leakage of individual key bits is present exclusively, it is reasonable to model and classify them separately. The observed probabilities of the key bits are, thus, independent and key enumeration can be performed in a straight-forward manner. Note that this means that there would be no overlap between the leakage $\mathcal{L}_j$ of individual bits. However, due

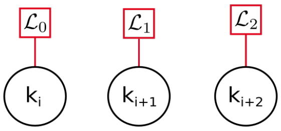


Figure 1: Leakage model with every bit leaking its value independently.

to the nature of high-frequency sampling of physical dimensions in side-channel analysis, an imminent low-pass behavior, thus, temporal overlap, should be expected for adjacent time-samples in many cases.

3.1.2 Overlapping value leakage

Figure 2 depicts a more realistic case, where the leakage signals of multiple key bits overlap temporarily. Hence, the leakage of individual key bits is observed at the same time and an attacker may only observe the distribution of a function of these bits. Signals from one variable do appear as switching noise in regard to the other variables (see Mangard et al. [MOP08] for a definition).

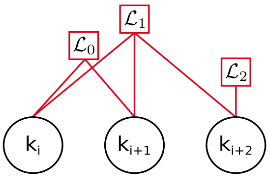


Figure 2: Leakage model with the leakage of bits overlapping in the observation.

Joint modeling of bits with multi-bit templates. In this case of overlapping leakage, it is reasonable to model those bits with overlapping signals jointly to reduce switching noise and increase the signal-to-noise ratio (SNR). Practically, this means that those multiple bits are 'put into the same template'. A difficulty arises when too many bits overlap so that they cannot feasibly be modeled jointly. One should then aim for a reasonable trade-off between the maximal number of values that can be profiled together, limited by computational and storage capabilities, the number of available traces for profiling of each class, as well as the SNR gain from reducing switching noise (see Section 4.2.1) and choose a suitable partitioning. Importantly, in this leakage model, the observed variables (bit values) are independent. Hence, key enumeration can be performed in a straight-forward manner using the derived probabilities for key bit values.

3.1.3 XOR leakage

Based on the DES key scheduling algorithm and general hardware design knowledge, one can assume that the *transition* between bits leads to observable leakage whenever they are consecutively written into storage cells. For instance, based on the assumption that consecutive round keys are written into a round key register, specific key bits follow each other on the respective storage cells. (Section 4 describes this in the context of the DES key schedule.) This *XOR leakage* implies that an attacker may observe the XOR difference between consecutive bits. Additional information about key bit values, i.e. value leakage, may also be observable. Masking of successive subkeys with the same mask throughout the key-schedule would for example result in pure XOR leakage. Wagner et al. [WHG17] assume that this is the case in their investigations. We actually prove this assumption to be true for the device under test described in Section 5.3. The case of XOR leakage is depicted in Figure 3.

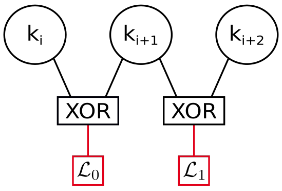


Figure 3: Leakage model with transition / XOR leakage of two bits.

Importantly, if XOR leakage between bits is observed, any derived (posterior or conditional) probabilities (i.e. classification results) for the key bit values are mutually dependent. Consequently, if an attempt is made to derive information about actual bit values (i.e. calculate marginal distributions for the key bits), information is lost due to their dependency. Described in simple terms, due to the observed XOR relations, the value of one bit depends on the other which does not allow to use most established key enumeration algorithms. Instead, Li et al.'s proposal [LWWW17] must be considered; see Section A.5.

Classifying XOR transitions instead of bit values. However, a different approach can be used. The observable variables, which are XORs of successive bits, are in fact *independent*. This follows from the fact that we can observe them directly and a priori the XORs of successive key-bits are uniformly distributed. Given the value of the first bit, there is a bijection between the other key bits and the succession of XORs (using the recurrence relation $k_i = (k_i \oplus k_{i-1}) \oplus k_{i-1}$ for successive key bits $\{k_i\}$). Since the key bits are independent, this implies the uniform distribution of the observed variables. Hence, successive XORs of key bits are mutually independent.

For the case of exclusive XOR leakage, the enumeration algorithm should be applied directly to the leaking XOR variables instead of the key bit values. This leads to a list of candidates for the transition sequence, where each entry represents two possible key candidates. We use this approach in our empirical study in Section 5 after confirming this leakage model. The discussion regarding temporal overlap of leakage from the previous Section 3.1.2 also applies here.

3.1.4 Mixed leakage

Finally, Figure 4 depicts a case, where both previously described forms of leakage appear in a device. Additionally, the depicted leakages $\mathcal{L}_j$ may also overlap. In this case the difficulty is that information about the bits are derived from two mutually dependent sources (value leakage and XOR leakage). Hence, a suitable algorithm is required to resolve this situation. Current literature [VCGS14, GS14] suggests that belief propagation on a factor graph similar to Figure 4 is the best choice for dependent variables.

Multi-bit templates. In the case of multiple kinds of leakage with possible temporal overlap, a careful grouping of variables into templates may increase SNR. Then, the marginalization to recover probabilities for subkeys would be done in either of the two directions; key bits, or observable XOR transitions. In this situation any marginalization leads to loss of information (e.g. for the XOR leakage part), since (a posteriori) there are no entirely independent variables. In practice, one needs to group variables with the strongest dependencies into the same templates and maximize the number of variables in each template, thus minimizing (but not preventing) information loss.

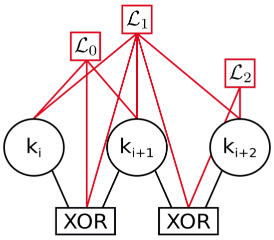


Figure 4: Leakage model with distance and value leakage.

Does it make sense to use overlapping templates? Overlap means that the same variables are profiled in more than one template in order to capture all available leakage during profiling. This could apply to pure XOR or mixed leakage. The downside of such an approach are resulting dependencies between subkeys, which prevent the use of classical key-enumeration algorithms. However, belief propagation based on the functional dependencies between the observed variables seems to be the reasonable approach to resolve dependencies stemming from overlaps. In practice, one has to carefully weigh the possible gain of information against the increased computational cost.

3.2 Distinguishing XOR and value leakage

For actual implementations, it is reasonable to either assume key bits are leaking their values directly, or to assume leakage of XOR transitions between key bits. Which leakage model applies can be tested by calculating exclusive SNRs for those leakage models.

Value leakage in every time-sample can be determined easily by computing SNR traces for individual key bits. If significant SNR is detected, the value leakage can be exploited directly in a multivariate template attack. XOR leakage is more difficult to assess. The following description focuses on two bits a and b where the value and XOR leakage is determined for the four different classes of the tuple (a, b). For two bits a and b, we first model leakage traces $X_{a,b}$ as univariate Gaussian distributions for all values of the tuple $(a, b) \in \{0, 1\}^2$ at time-samples $t \geq 0$ as

$$X_{a,b}(t) \sim \mathcal{N}(\mu_{a,b}(t), \sigma^2_{a,b}(t)) \qquad \text{for all } a, b \in \{0, 1\}. \tag{1}$$

The number of traces in every class is $N_{a,b}$. The distributions of the traces X_a and X_b of individual bits a and b are again profiled as Gaussian and characterized through means and variances μ_a, σ_a and μ_b, σ_b respectively (formula for $a = 0$, other cases analogous; valid if $N_{a,b} \simeq N/4$, where N is the total number of traces). The distributions are an approximation of the mixture of two Gaussians, which has moments

$$\mu_{a=0}(t) = \frac{1}{2}\left(\mu_{a=0,b=0}(t) + \mu_{a=0,b=1}(t)\right) \text{ and} \tag{2}$$

$$\sigma^2_{a=0}(t) = \frac{1}{2}\left(\sigma^2_{a=0,b=0}(t) + \sigma^2_{a=0,b=1}(t)\right) + \frac{1}{4}\left(\mu_{a=0,b=0}(t) - \mu_{a=0,b=1}(t)\right)^2. \tag{3}$$

The SNR of variable a can be computed as

$$\mathrm{SNR}^{value}_a(t) = \frac{\mathsf{Var}[\mu_a(t)]}{\mathsf{E}[\sigma^2_a(t)]}. \tag{4}$$

The XOR leakage can be determined by comparison of joint and single-bit SNR traces. First, the SNR trace for two joint key bits $\mathrm{SNR}^{\mathrm{total}}(t)$ is computed. This contains all leakage from bit values and their XOR transitions combined.

$$\mathrm{SNR}^{\mathrm{total}}(t) = \frac{\mathsf{Var}[\mu_{a,b}(t)]}{\mathsf{E}[\sigma^2_{a,b}(t)]}. \tag{5}$$

Then, the SNR of pure value leakage is calculated from a combination of SNRs of the individual bits, while assuming their mutual independence and independent noise. Based on the above described value SNR, the combined value leakage can be computed as

$$\mathrm{SNR}^{\mathrm{value}}_{a,b\,\mathrm{combined}}(t) = \frac{\mathsf{Var}[\mu_a(t)] + \mathsf{Var}[\mu_b(t)]}{\sigma^2_{\mathrm{pooled}}(t)}, \tag{6}$$

with

$$\sigma^2_{\mathrm{pooled}}(t) = \frac{1}{2}\left(\mathsf{E}[\sigma^2_a(t)] + \mathsf{E}[\sigma^2_b(t)]\right). \tag{7}$$

If both, $\mathrm{SNR}^{\mathrm{total}}(t)$ and $\mathrm{SNR}^{\mathrm{value}}_{a,b\,\mathrm{combined}}(t)$ are equal, no additional information from the XOR is available that exceeds the information derived from the individual bits. Hence, for a model to have additional XOR leakage (or any other leakage that is caused by variables that depend on multiple bits), $\mathrm{SNR}^{\mathrm{total}}(t)$ must be higher than $\mathrm{SNR}^{\mathrm{value}}_{a,b\,\mathrm{combined}}(t)$. For instance if masking is implemented and both successive values are masked using the same bit-mask, the attacker is indeed unable to distinguish the values of single bits ($\mathrm{SNR}^{\mathrm{value}}_{a,b\,\mathrm{combined}}(t) = 0$), but could still observe XOR leakage.

4 DES key schedule and templates for implementations with exclusive XOR leakage

The DES algorithm has a comparably simple key scheduling algorithm. After removing the parity bits from the original 64 bit DES key, 56 effective key bits remain. Those are divided into two halves of 28 bit each, handled in registers C and D (notation according to specification). Round keys are generated through rotating those halves by one or two bits to the left (depending on the round), and using a so-called permuted choice to select 24 bits out of each 28 bit register. All round keys are, hence, a permuted selection of bits from the original key. An initial *permuted choice 1* leads to a first round key already permuted from the original one. Figures 5a and 5b depict the selection of bits for the 16 round keys (indices applied after removing parity bits). Each key bit is used approximately in 14 out of 16 rounds. Whenever a key bit is handled, exploitable side-channel leakage might occur, i.e. bit values or the XOR between bits depending on the implementation's countermeasures. Two bit pairs are highlighted in red and blue to illustrate that due to the schedule, even XOR transitions between specific bits re-occur which increases exploitable leakage of the respective transitions.

4.1 Leakage of the DES key schedule

It depends on the implementation, which kind of leakage is observable. Wagner et al. (see Section 2) assume that their attacked implementation stores successive round keys in dedicated round key registers. Hence, in these registers, successive round keys generate leakage related to the transition between bits. Further, Wagner et al. observe, that their implementation seems to leak the XOR of those transitions which could be explained by a constant masking (note that they did not actually confirm this assumption).

When looking at successive round keys in Figures 5a and 5b, one finds that certain bits always follow on certain other bits. For example, at a certain position in the registers,

Round key #	Indices refer to the input key excluding parity bits																						
0	8	44	29	52	42	14	28	49	1	7	16	36	2	30	22	21	38	50	51	0	31	23	15	35
1	1	37	22	45	35	7	21	42	51	0	9	29	52	23	15	14	31	43	44	50	49	16	8	28
2	44	23	8	31	21	50	7	28	37	43	52	15	38	9	1	0	42	29	30	36	35	2	51	14
3	30	9	51	42	7	36	50	14	23	29	38	1	49	52	44	43	28	15	16	22	21	45	37	0
4	16	52	37	28	50	22	36	0	9	15	49	44	35	38	30	29	14	1	2	8	7	31	23	43
5	2	38	23	14	36	8	22	43	52	1	35	30	21	49	16	15	0	44	45	51	50	42	9	29
6	45	49	9	0	22	51	8	29	38	44	21	16	7	35	2	1	43	30	31	37	36	28	52	15
7	31	35	52	43	8	37	51	15	49	30	7	2	50	21	45	44	29	16	42	23	22	14	38	1
8	49	28	45	36	1	30	44	8	42	23	0	52	43	14	38	37	22	9	35	16	15	7	31	51
9	35	14	31	22	44	16	30	51	28	9	43	38	29	0	49	23	8	52	21	2	1	50	42	37
10	21	0	42	8	30	2	16	37	14	52	29	49	15	43	35	9	51	38	7	45	44	36	28	23
11	7	43	28	51	16	45	2	23	0	38	15	35	1	29	21	52	37	49	50	31	30	22	14	9
12	50	29	14	37	2	31	45	9	43	49	1	21	44	15	7	38	23	35	36	42	16	8	0	52
13	36	15	0	23	45	42	31	52	29	35	44	7	30	1	50	49	9	21	22	28	2	51	43	38
14	22	1	43	9	31	28	42	38	15	21	30	50	16	44	36	35	52	7	8	14	45	37	29	49
15	15	51	36	2	49	21	35	31	8	14	23	43	9	37	29	28	45	0	1	7	38	30	22	42

(a) Round keys: Register C.

Round key #	Indices refer to the input key excluding parity bits																						
0	19	24	34	47	32	3	41	26	4	46	20	25	53	18	33	55	13	17	39	12	11	54	48	27
1	12	17	27	40	25	55	34	19	24	39	13	18	46	11	26	48	6	10	32	5	4	47	41	20
2	53	3	13	26	11	41	20	5	10	25	54	4	32	24	12	34	47	55	18	46	17	33	27	6
3	39	48	54	12	24	27	6	46	55	11	40	17	18	10	53	20	33	41	4	32	3	19	13	47
4	25	34	40	53	10	13	47	32	41	24	26	3	4	55	39	6	19	27	17	18	48	5	54	33
5	11	20	26	39	55	54	33	18	27	10	12	48	17	41	25	47	5	13	3	4	34	46	40	19
6	24	6	12	25	41	40	19	4	13	55	53	34	3	27	11	33	46	54	48	17	20	32	26	5
7	10	47	53	11	27	26	5	17	54	41	39	20	48	13	24	19	32	40	34	3	6	18	12	46
8	3	40	46	4	20	19	53	10	47	34	32	13	41	6	17	12	25	33	27	55	54	11	5	39
9	48	26	32	17	6	5	39	55	33	20	18	54	27	47	3	53	11	19	13	41	40	24	46	25
10	34	12	18	3	47	46	25	41	19	6	4	40	13	33	48	39	24	5	54	27	26	10	32	11
11	20	53	4	48	33	32	11	27	5	47	17	26	54	19	34	25	10	46	40	13	12	55	18	24
12	6	39	17	34	19	18	24	13	46	33	3	12	40	5	20	11	55	32	26	54	53	41	4	10
13	47	25	3	20	5	4	10	54	32	19	48	53	26	46	6	24	41	18	12	40	39	27	17	55
14	33	11	48	6	46	17	55	40	18	5	34	39	12	32	47	10	27	4	53	26	25	13	3	41
15	26	4	41	54	39	10	48	33	11	53	27	32	5	25	40	3	20	24	46	19	18	6	55	34

(b) Round keys: Register D.

Figure 5: DES round keys in parts C and D. Two bit pairs are highlighted in red and blue to illustrate the reoccurrence of transitions.

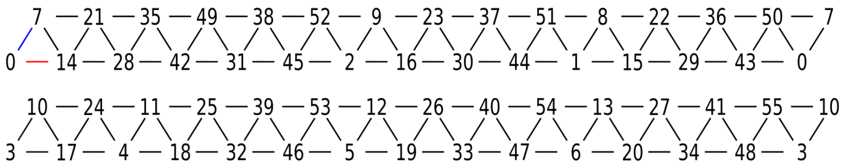


Figure 6: Observable transitions for key bits in register C (top) and D (bottom).

the bit with index 0 is always preceded either by bit 7 or by bit 14. The mentioned bits are marked in Figure 5a. Figure 6 depicts these relations as two groups of bits from the two registers C and D. The figure describes all possible transitions between key bits, that could lead to exploitable leakage, given the assumption that the round key register is implemented as described. The number of actually observed transitions is small compared to the theoretical maximum of key bit combinations, which stems from the DES algorithm. It can be observed that some transitions occur more frequent than others. The transition between bits 0 and 7 is marked in blue in Figure 5a and occurs three times, while the transition between bits 0 and 14 is marked in red and occurs 10 times.

Figure 7 depicts a matrix of transitions between bits. Bit numbers on both axis represent bits before (x-axis), and after (y-axis) round-key updates. The color represents the number of rounds that each transition is part of. Rare transitions occur 3 times at maximum, others occur 12 times at minimum. Every key bit is involved in a higher-frequent and a lower-frequent transition, which we call *strong* and *weak* transitions. For side-channel analysis, this means that most transitions are observable in multiple rounds for exploitation.

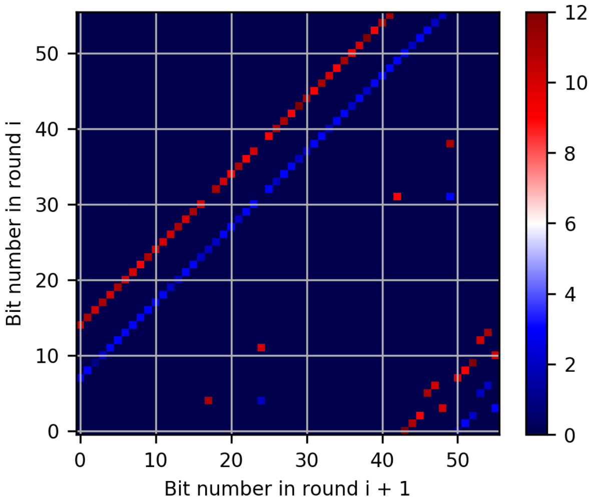


Figure 7: Matrix showing the occurrence frequency of transitions between successive round key bits. Strong transitions in red, weak transitions in blue.

The amount of leaking information in each measurement is different, depending on the number of re-occurrence. When grouping all bits and transitions which are connected, one obtains the relations shown in Figure 6.

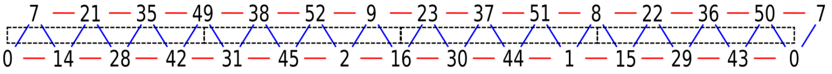


Figure 8: Transitions for key bits in C register. Coloring depicts occurrence frequency. The dashed boxes mark the XORs used to build unique labels for the four subkeys.

Figure 8 depicts half of the transitions from Fig. 6, with a coloring depicting the frequency of occurrence similar to as in Figure 7.

4.2 Templates for exclusive XOR leakage

If an implementation leaks only the XOR between bits, it makes sense to profile and target this XOR leakage directly. This is argued in Section 3.1.3. Figure 8 depicts all transitions for half of the key. Since register updates of round keys happen simultaneously, *the leakages of all individual transitions overlap.* The question now is, which XORs should be profiled together to create multi-bit (or rather multi-XOR) templates. The discussion in Section 3.1.3 is applicable here. It makes sense to combine XOR transitions which share bits into the same template. Figure 8 depicts black boxes around 4 groups of 7 XOR transitions. This is an example for choosing a template size of 7 bit. It means that as template labels, not the bits themselves are used, but the XOR between the bits. This creates unique labels for all observable classes. At first glance it seems, that such templates would only cover the less occurring transitions marked in blue (as included by the black dashed boxes in Figure 8). However, the other ones are implicitly included as well. For example, the values of $k_0 \oplus k_7$ and $k_7 \oplus k_{14}$ determines $k_0 \oplus k_{14} = (k_0 \oplus k_7) \oplus (k_7 \oplus k_{14})$. Hence, the transitions marked in red are covered since there is only ever one possible red transition for the included blue transitions. The blue transitions can be used to uniquely label all classes of XOR leakage.

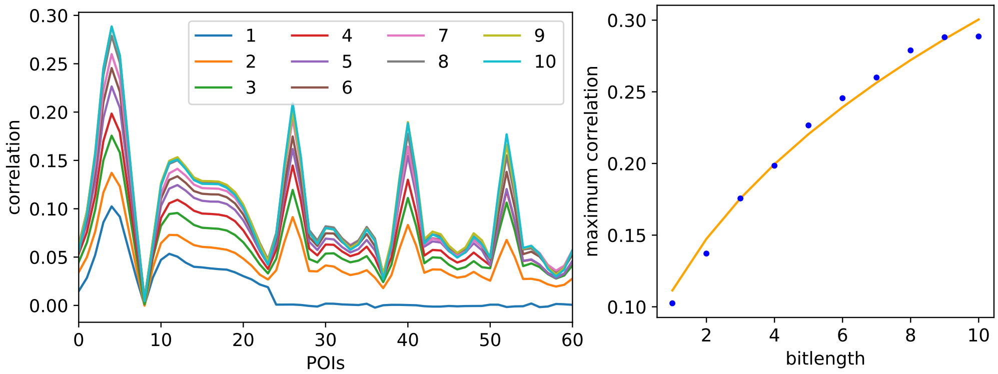
We chose a template size of 7 XOR transitions in the practical investigation in Section 5.

Note that in order to derive the value/label of the XOR during profiling for 7 XOR transitions, 8 key bits are involved. For such 7 bit XOR templates, 4 templates cover one of the two registers C, or D. Hence, a total number of 8 templates covers all key bits, respectively XOR transitions. To be precise, of the 28 successive XORs, only 27 are independent. The last XOR can be recovered from the first 27.[3] Therefore the last template for each register effectively only has six bit, totaling 54 recoverable bits. After recovering all XORs there is one bit of choice left in both registers to choose an actual value based on the XORs. Thus, two more bits need to be added to the final security level after applying the key rank algorithm. These two bits are inherently undetermined due to the XOR leakage and, thus, the minimal possible remaining entropy for this leakage model is always two bit.

Strictly speaking, not 100% of the red transitions are covered. Few transitions between adjacent bits from different templates are disregarded, specifically, one red transition at every template boundary. For the case of 7 bit templates, there are 4 boundaries for every 28 bit register, thus, there are 4 disregarded red transitions. As a rough estimation, this omission of 4 out of 56 observable transitions per register equals disregarding 7% of the available leakage. This could be prevented by overlapping templates covering one more blue transition to also cover the previously disregarded red one. After classification, this overlapping bit could be omitted again to go into key enumeration with independent XOR classifications. Larger templates, i.e. 9 or 10 bit templates, lead to less template boundaries. However, the relative improvement is small compared to the computational overhead and increased measurement complexity. Given all facts, we decided not to use overlapping templates, to avoid complexity while accepting this limited information loss.

Note that Wagner et al. use overlapping templates for a different reason, because they (unfortunately) model bit values in their templates and want to connect adjacent templates despite the XOR leakage. As described in Section 3.1, this approach of modeling bit values is not suitable for XOR leakage.

4.2.1 On the size of templates



(a) Correlations for $b \in \{1, \dots, 10\}$ over POIs

(b) Maximal correlations

Figure 9: CPOI results on measured traces. Template bit lengths b between 1 and 10.

Technically, larger templates allow a reduction in encountered switching noise from the remaining key bits. This effect is shown on the correlations calculated using CPOI [DS16] (see Section A.2.3) on measured traces from the empirical evaluation in Section 5 for different templates bit lengths b in Figure 9. On the left, the correlations are shown for the

[3] For each 28 bit register, we choose the value of one ’first’ bit along the chain, e.g. b_0 (see Fig. 8). Then the values of all 27 other bits b_i are determined by 27 XORs. The last XOR only completes the cycle.

first 60 POIs of the traces. On the right, the maximal correlation is plotted over the bit length of the templates. The fit (orange) shows the theoretical dependence of correlations on b, if each bit has independent and identically distributed Gaussian leakage. It can be noted that for higher numbers of bits the gain is limited. The penalty in computation and required traces, however, is significant, as the matrix inversion alone scales at least as $\mathcal{O}(2^{b \cdot 2,373})$, depending on the algorithm. Furthermore, the profiling set of a template attack has to double in size for each extra bit to achieve a similar estimation error.

4.3 Comparison to AES

AES uses a more elaborate key scheduling algorithm with a non-linear substitution function. This means that specific parts/bits of the original key appear only in the side-channel of the first one/two round keys for exploitation through templates. Full diffusion is reached after two rounds. Intermediate values of the key schedule which are connected in the algorithm, but in a non-linear way, need to be classified independently and more complex strategies for exploitation of many values need to be used.

5 Empirical study: Security controller

We performed a profiled template attack against the DES key-schedule executed by a security controller. Although we were not able to perform measurements on the very same device as the one used by Wagner et al. due to availability issues, we have very carefully chosen our DUT to be as closely matched as possible. In particular we have strong evidence that the underlying hardware architecture is identical.

5.1 Chip preparation, measurement setup, and alignment

The DUT was decapsulated and thinned to perform measurements from the backside of the chip. The measurement setup is described in Section A.6. After a first test using the 500 µm diameter EM probe, we proceeded to use a 150 µm diameter probe for the main measurements. This choice is backed by previous experience and led to useful results in this study. The sampling rate was 5 GS/s.

5.1.1 Identifying the DES execution

The DES operation was called using different lengths for the plaintext as multiples of the block size. A current measurement and an EM measurement were used to visually inspect the operation of the controller. The 500 µm EM probe was positioned in a region, where varying patterns were visible during the whole time of operation. The time of the DES execution could be narrowed down to ≈ 100 µs because traces show a distinct and different pattern during this frame. The length of the pattern multiplies according to the number of input blocks, i.e. DES encryptions. A DES operation is expected to take ≈ 1 µs for roughly 16 cycles at an internal clock frequency, which should be around 30 MHz, as most internal clocks of current devices operate in this range. This means that the timing of the DES execution has to be further narrowed down.

Using the 500 µm diameter probe, we manually selected different positions while inspecting the EM traces. In one specific area, we observed a general low activity with the exception of a short time of high activity and large peaks within the previously identified time frame. This section is depicted in Figure 10 and fits the assumption of a hardware core, which is mostly inactive, and activates only during DES executions. The execution spans $\approx 0.9 \mu s$ (4400 samples at 5 GS/S) as expected. The clock cycles are visible as peaks and indicate a clock frequency of ≈ 32.5 MHz.

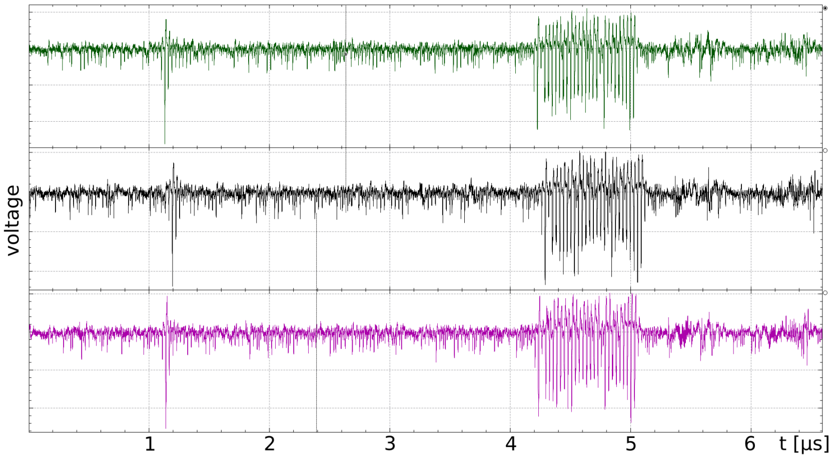


Figure 10: Three EM traces showing DES executions at a manually selected position measured with a 500 µm probe.

5.1.2 Alignment

We use a two-stage triggering in the oscilloscope with a first and second trigger on the communication line monitoring the commands sent to and from the smartcard. No additional trigger on the measured current is required, since the smartcard software execution is largely deterministic and nearly constant in time. Three executions recorded in this manner are shown in Figure 10. It is evident that the triggering leads to a good initial alignment.

The device likely uses an internal clock source, which is not synchronized to the oscilloscope clock. Hence, some local misalignment through drift and jitter is expected which needs to be removed during post-processing. See Section A.1 for a background on the discussed alignment methods. For the main analysis, we only use static alignment to achieve the best possible local alignment. First, we use a short trace sample for least square error matching. Then, we cut the trace into segments, each containing one clock cycle. Given the large peaks in the measurements, we can align each segment using the edge of the peak.

Elastic alignment based on dynamic time warping (Section A.1) was used only during initial tests to quickly identify or rule out leakage in a new set of traces. It was not used on trace sets prepared for template attacks, because it led to slightly worse local alignment and lower correlation coefficients in the leakage test.

5.1.3 Comparing the number of DES executions to other implementations

Figure 11 shows a zoomed view of a DES execution including leakage test results as described in the following Section 5.2. The DES engine performs 16 single-cycle rounds followed by eight rounds of backward computation, most likely for the purpose of protection against fault attacks. This assumption is confirmed by the peaks in the correlation trace of the CPOI leakage test (Section A.2). Thus, each trace contains a single DES execution and eight rounds of backward computation.

Wagner et al. observe four DES executions with 16 peaks in each single trace of their *single trace attacks*. We attribute this deviation to a different software stack on their device. Note that although the attack is mounted solely on leakage of the (presumable identical) hardware crypto engine, this likely influences the exploitability due to differing numbers of available hardware executions per recordable trace. Wagner et al. assume that the four DES executions are due to a countermeasure against differential fault analysis

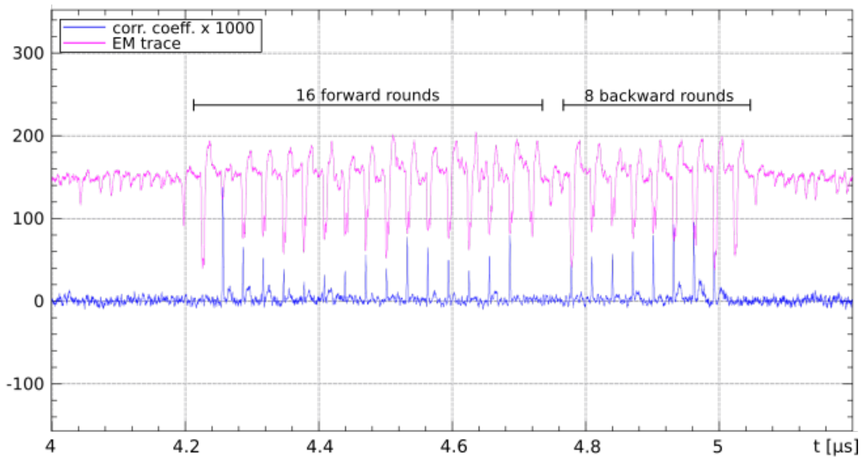


Figure 11: Zoom into EM measurement of one DES execution at a manually selected position measured with a 500 µm diameter probe is shown in pink. The correlation coefficient of a CPOI leakage test for 7 bit XOR templates is shown in blue, scaled up by a factor of 1000 (values ranging up to ≈ 0.15). The DES rounds are marked.

(DFA). We suggest that its purpose is to ensure the upward compatibility of a triple-DES implementation with single DES [MH81]. In this way, a triple-DES implementation (i.e. software interface) would use the same key for subsequent encryption, decryption and encryption. Only the fourth DES execution would be a DFA countermeasure. Importantly, this would mean that Wagner's results would not scale to triple-DES in the way described. Instead the attacks on the individual DES keys of a triple-DES would have less exploitable leakage (e.g. only one DES execution instead of four). We will show that the difference between one and slightly more DES executions per key results in a non-neglegible difference in the security level.

5.2 Leakage test, measurement position, and POIs

We first performed a CPOI leakage test on the Hamming distance of successive round keys. See Section A.2 for the background on all leakage tests used in this study. For that purpose, we assumed XOR leakage as described in Section 3.1.3. Note that an attacker would likely test value leakage (e.g. in a simple form a Hamming weight model) first, however, from Wagner's results we already know to prioritize XOR leakage. Traces were prepared as described in Section 5.1.

5.2.1 Initial leakage test

The key was split into eight subkeys, each covering seven XORs between successive key bits, as described in Section 4.2. The CPOI leakage test was performed on these 7 bit XOR templates. We used 200k traces acquired with a 500 µm diameter probe at a measurement position, which was selected manually as described above. The test produced positive results with correlation coefficients in the range of 0.1 to 0.2. For comparison, Wagner et al. report higher correlation coefficients up to 0.4 with an unknown probe (see Fig. 6 in [WH17]). Figure 11 shows the leakage trace of one subkey in blue (scaled up by a factor of 1000) together with an exemplary measurement trace in pink. As expected, we observe leakage at peaks where round keys are overwritten. No leakage is detected at the time of the first, last, and last backward computation peak. During the actual DES execution excluding the backward computation, 15 leakage peaks are observed as expected.

5.2.2 Determining a measurement position using the t-test

The 150 µm diameter probe is usually expected to give better results than the larger 500 µm diameter probe, but needs to be positioned more accurately. We performed measurements at many positions in a grid to find the optimal position. Every position was tested for leakage using a fixed vs. random t-test on the key. The use of a t-test instead of a CPOI leakage test and the relatively high leakage allowed us to use very few traces for this purpose. Initial tests using the 150 µm probe at a manually selected position close to the previous one revealed that relatively few traces are sufficient to detect leakage with a good margin of error. We scanned a part of the backside surface of the die centered around the manually identified position. We used a high resolution for the grid with 15 times 12 positions and steps of roughly 50 µm. This large number of positions is possible due to the low measurement and memory complexity of the t-test.

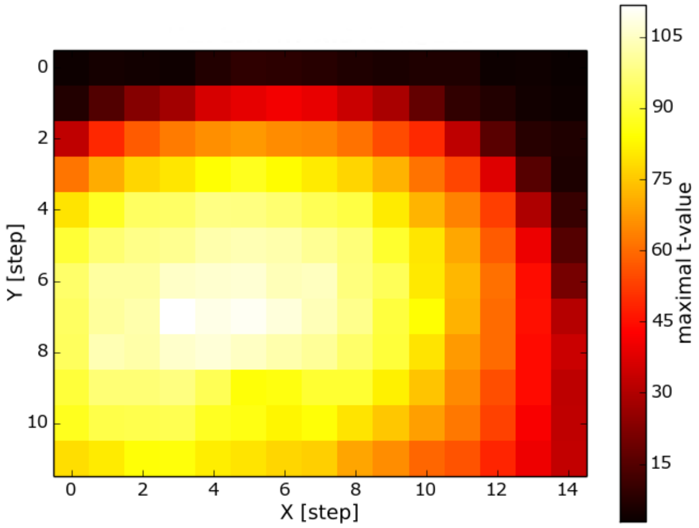


Figure 12: Maximum of t-value trace per position on a 50 µm grid using a 150 µm diameter probe.

Figure 12 depicts the resulting maximum t-value at all measurement positions. We chose the position with the highest value for all further analyses. Note that the selection of a measurement position based on the univariate maximum value does not guarantee the best results, since template attacks exploit multivariate leakage. However, the univariate t-test is computationally fast and represents the leakage of the whole key at once. The CPOI leakage tests for subkeys could indicate different optimal measurement positions for different subkeys. For practicality, we use only one position during all following measurements.

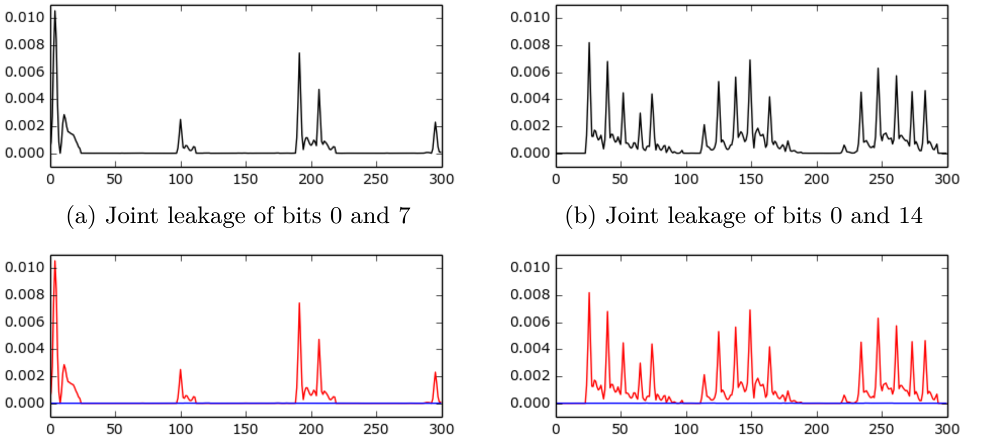
The alignment of traces at different positions requires an alignment strategy which is robust to expected differences between traces at different positions, in particular in the signal amplitude. Fortunately, static alignment could be achieved with low manual tuning efforts by a least squares matching using a pattern from a single trace and one position.

5.2.3 Determining POIs using a CPOI leakage test

We used the previously described segment-wise edge-alignment before performing a CPOI leakage test. Using the smaller probe at the new measurement position led to improved correlation coefficients in the range of 0.15 to 0.25. For further analyses we chose the 300 time-samples with the highest correlation coefficients in the CPOI leakage test as points-of-interest (POIs). No other or further trace compression is used. See Section A.3 for background on trace compression.

5.3 Leakage model

At the selected position and using 300 POIs we investigated the leakage model of the device in more detail. Even though the assumed XOR leakage led to positive results, this step is required to make the necessary decisions for the template attack (see descriptions in Section A.4 and Section 3).



(a) Joint leakage of bits 0 and 7

(b) Joint leakage of bits 0 and 14

(c) XOR-leakage (red), value leakage (blue)

(d) XOR-leakage (red), value leakage (blue)

Figure 13: SNR (y-axis) for the transition between key bits 0 and 7 on the left, and 0 and 14 on the right over 300 POIs (x-axis). The joint leakage is entirely caused by XOR transitions (red) while no value leakage (blue) is detectable.

We selected three exemplary bits from the key which share two transitions according to the DES key schedule, i.e. they follow each other at the same bit position in two subsequent round keys. Hence, there could be observable XOR leakage from the transitions of each bit pair. The question is, whether the observable leakage is determined by the value of the individual bits, or by the XORs between them. Figure 13 shows the computed SNR for key bits 0 and 7 and bits 0 and 14. Note that the strong transition between the two bits 0 and 14 occurs in ten out of 15 round key updates. The analyzed trace contains a total of 15 round key updates plus eight updates from the backwards computation. The weak transition between bits 0 and 7 occurs only in three round-key updates per DES execution. The figure shows the SNR of the joint leakage for both bit pairs in the upper graphs. There is no value leakage from individual bits when profiled separately, as shown in the lower graphs in blue. The presence of value leakage would indicate an additional source of leakage, for example from an unmasked bus. The entire leakage is caused by the transition between the two bits. The corresponding XOR leakage is shown in the lower graph in red. *We conclude that the device exhibits XOR leakage and no value leakage.*

Additionally, we found that the XOR leakage for different bit-pairs has different SNRs, which is not entirely explained by the number of rounds they appear in. This could be caused by properties of the hardware-implementation and possibly depends on the chosen measurement position, and is expected.

5.4 Evaluating attack success

Profiled template attacks and successive key enumeration yield a list of key candidates in order of likelihood. The position of the unknown secret key k in the list is called the key rank $R(k)$. In our situation the attacker is unlikely to retrieve the correct key as the first entry in the list and the key rank should be treated as a random variable dependent on input data and noise. The remaining brute-force effort to find the secret key after a template attack can be expressed through the expected key rank $\mathsf{E}[R(k)]$, which is called guessing entropy in the literature. We find that our results are best represented by the logarithm of the key rank, which we call the *security level* $S(k) = \log_2 R(k)$. The expected security level $\mathsf{E}[S(k)]$ is a good measure to compare outcomes of different attacks, although it does not directly correspond to the brute-force complexity, since expectation and logarithm do not commute. It was introduced under the name ranking entropy in Martin et al. [PMMOS16]. They also have an in depth discussion on the presentation and comparison of their side-channel results, which resemble ours. Similar to their findings, we chose to state attack outcomes more precisely in terms of percentiles of the empirical distributions for the security levels.

Several other metrics designed for security evaluation can be found in literature. A security graph [VCGS13] is a popular tool to assess the security of implementations or devices after SCA. It relates the number of traces available for the attack (x-axis) to the achieved reduction of security described by the minimum, maximum and average rank of the entire key. It displays the distributions of key ranks and thus brute-force complexity for each possible number of traces. The security graph is well suited for DPA, where the attacker has to manage this trade-off in his favor. In case of DPA, the attacker usually has the necessary choices regardless of the actual device or key. The graph allows an attacker to record the optimal number of traces such that the required brute-force effort matches the computational capabilities. In our case, however, the success rate of an attack does not strictly increase with more traces. After a relatively small amount of traces (e.g. ≈ 400) further improvement is hindered by noise factors that do not average-out. Hence, it seems most reasonable to assess the security based on this limit-distribution, which we approximate using 900 traces. For each device the achievable security reduction is predetermined by the key and unknown to the attacker. The attacker can improve the achievable security level only by changing the device after each unsuccessful attack, not by increasing the number of traces.

Two other measures for the security of a device have been used in related work. The n-th order success rate [SMY09] is defined as the percentage of cases where all subkeys are ranked among the first n ones over the number of used traces. In its original form, the guessing entropy [KB07] describes the average rank of subkeys over the number of used attack traces. Applied to the entire key, it is equivalent to the average key rank. Any metric that is purely based on the ranks of subkeys is outdated and not suitable to describe the effort of enumerating candidates for an entire key.

5.5 Results from profiled template attacks

The template attacks are performed as described in Section 4.1. Additional details can be found in Section A.4. At the selected measurement position, we recorded 2.5 million profiling traces with random keys and inputs. For the attack, we used 1k random keys and record 1000 measurements with random inputs for every key, from which we use 900 to allow for dismissal of traces which cannot be aligned properly. We present results for template attacks with 1, 3, and 900 traces per key. The attack results are mainly affected by electrical noise and switching noise from unprofiled parts of the key and varying plaintexts. By using more traces the influence of electrical noise and switching noise from varying plaintexts is averaged out, which improves results. However, switching noise from unprofiled parts of the key cannot be removed since the leakage of the key-schedule does not depend on any variable inputs.

For the profiling phase of the template attack, we profiled the *XOR distances* of the subkeys instead of the bit values, as described in Section 3.1.3. Each attack on a subkey results in a list of candidates with corresponding a-posteriori probabilities. Key enumeration was performed directly on XOR candidates. The fact that the classified variables are *independent* allowed us to employ *optimal key enumeration algorithms*. Hence, there is no information loss from making compromises regarding the chosen algorithms and their requirements. The only information loss stems from dividing the key into subkeys, which is necessary as the full 28 bit per register cannot be profiled at once. By profiling subkeys independently of each other, the remaining key parts appear as switching noise. Further, a total of three transitions are missing in the templates as described in Section 4.2. We did not observe any clear improvement of results from using overlapping templates that cover the missing transitions, as discussed briefly in Section 5.6.1. The entire key is determined by the sequence of XOR values and the remaining choice of the first bit for each register respectively. Hence, to obtain the final security level, two bits are added to the output of the key rank algorithm to account for the remaining possible choices when recovering a key from the XOR transitions.

5.5.1 Security levels exhibit wide distributions

For most symmetric ciphers, for example when performing DPA attacks, the achieved security level is largely independent of the actual key value. The following results show that the security levels vary, even when attacking keys with a high number of traces to minimize the influence of electrical noise. We estimate security levels for each attacked key for three different cases, using 1, 3 and 900 traces.

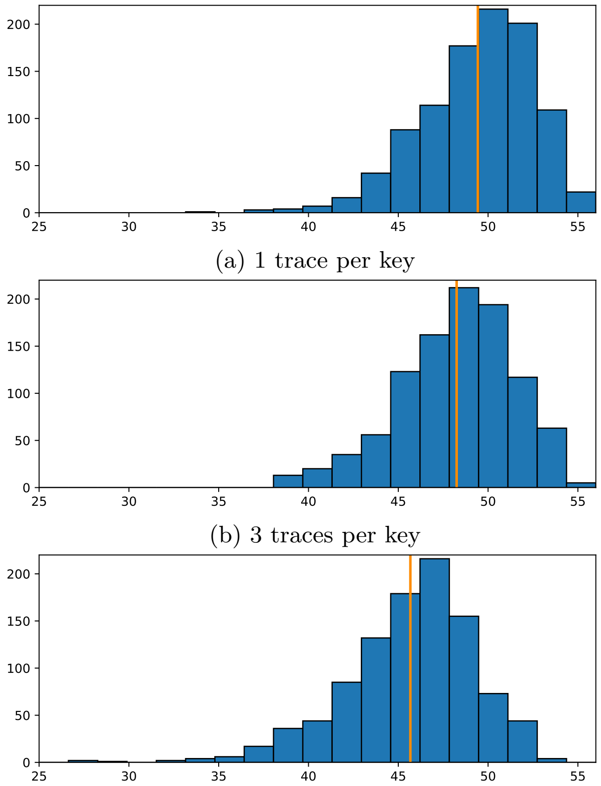
Figures 14a, 14b, 14c depict histograms of the resulting security levels. The DES executions include eight additional and redundant rounds of backward computation. Our case of three attacked traces, hence, roughly corresponds to the case described by Wagner et al., where a single measurement includes four DES executions.

The security levels for different keys vary and include results as low as 25 *bit (out of 1k keys).* Testing more random keys will most likely reveal additional keys with very low security levels in the tail of the distribution. As expected, the average security level decreases when more traces are used for the attack, since the influence of noise is reduced. Hence, results after 900 attack traces exhibit lower security levels than after one or three traces. Apart from electrical noise, also the part of the switching noise which is caused by changing plaintexts averages out.

Table 1: Average security levels after attacking a varying numbers of traces per key.

	This work			**Wagner et al.**
	1k keys, 300 POIs			297k keys, 352 POIs
	1 trace	3 traces	900 traces	1 trace
	1.5×DES per trace			4×DES per trace
Mean [bit]	49.4	**48.2**	45.7	**46.16** [WHG17, Fig. 11]

Table 1 lists the mean security level for the three cases after an attack with 300 POIs. The number of DES executions per trace is an important factor in the achievable entropy reduction. The average security level after a *three trace attack* on our DUT significantly lowered and amounts to 48.2 bit, which is higher than the results of Wagner et al. at 46.16 bit (cf. Fig. 11, [WHG17]). We also performed the attacks using 900 instead of 300 POIs, which increases runtime significantly. Average security levels are approximately 2 bit lower.



(a) 1 trace per key

(b) 3 traces per key

(c) 900 traces per key

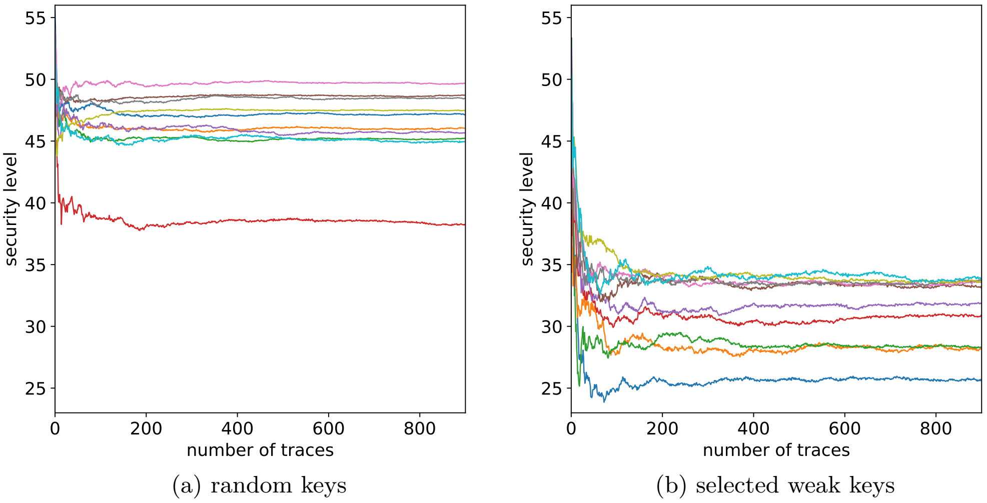
Figure 14: Histograms of security levels after template attacks on 1000 keys. Security levels are plotted on the x-axis, number of keys per bin on the y-axis. Results are shown for 300 POIs and three different numbers of attack traces (1,3 and 900) per key. The mean security level is marked by an orange line.

In addition, we tested two keys with key bits all zeros or all ones, which represent the worst case as their traces have a unique profile (assuming perfect XOR leakage). Note that those keys are also cryptographically weak and unrealistic in practice. *Cryptographically weak keys measured on the device with 900 traces per key have security levels of* 2 *bit.* Increasing the number of tested random keys will increase the number of observations with exceptionally low security levels (not necessarily as low as 2 bit).

5.5.2 Security levels converge key-dependent in the low noise limit

What remains is to find the *reason* for the distribution of security levels. In this part we show that noise from insufficient sample sizes cannot explain the observations.

Figure 15 shows the security levels of selected keys over an increasing number of traces. The left part of Figure 15 depicts randomly chosen keys, the right side shows keys that were selected for their low security levels. *The security levels of keys converge to different limits as the number of attack traces increases.* The limit is reached (up to a small error) after roughly 200-400 traces. The histogram in Figure 14c for an attack with 900 traces is therefore close to the limiting distribution. The spread of the security levels is not caused by an insufficient number of attack traces. This rules out electrical and switching noise of varying plaintexts on the attack traces as reason for the variation. Note that attacks with few traces can lead to security levels below the low noise limit, which is due to beneficial noise circumstances. An attacker is, however, of course unable to deliberately select such



(a) random keys

(b) selected weak keys

Figure 15: Dependence of the security levels of chosen keys on the number of used attack traces. *Left:* Ten randomly selected keys. *Right:* Ten selected keys with low security levels.

measurements. Hence, the low noise limits of the security levels best describe what an attacker can expect.

We ruled out two other possible influences as reason for the observations. First, we tested for statistical artifacts caused by too low numbers of traces in the profiling set, which could lead to a bias. We repeated the profiling measurement (using the same amount of measurements) and achieved identical attack results with the new profiling set. Second, we studied the influence of the measurement position by repeating all measurements at a different position. All results are comparable to those presented in this paper.

We conclude that keys have inherent security levels which are distributed and include cases of very low security levels.

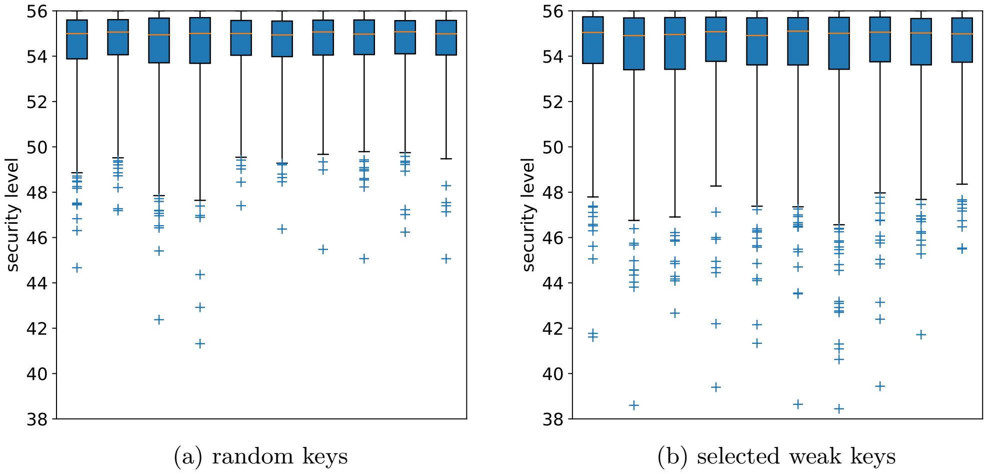
5.5.3 Influence of noise on single trace attacks is high

Even though keys possess inherently different security levels, noise has a high influence on individual attack results. In the following, we demonstrate this by performing a large number of single trace attacks against the same selection of keys, shown in Figure 15. Instead of computing security levels after attacking multiple traces and combining results, we compute security levels for 900 individual single trace attacks. Figure 16 shows a box plot of ten random keys on the left and the ten *weak* keys on the right. Somewhat surprisingly, given the results after 900 traces in Figure 15, every *weak* key exhibits a broad range of single trace attack results, most of which are close to chance level (55 bit). The weak keys on the right part of Figure 16 only lead to lower security levels more frequently and there are more outliers. *This means that the success of a single trace attack (or an attack with few traces) depends highly on noise.*

Attackers who are allowed only one measurement per key may repeat the attack on different keys or different devices. Every attack will lead to an expected security level result characterized by the distribution in Figure 14a. The consequences on practical attacks, in particular for triple-DES, are discussed in Section 6.

5.6 Generalization through 'noiseless' simulation

The previous investigations confirm that different keys can inherently be attributed to different security levels. In this section, we use a simplified simulation model to study the



(a) random keys

(b) selected weak keys

Figure 16: Box plot of resulting security levels from 900 single trace attacks for ten randomly selected keys on the left and ten weak keys on the right. The interquartile range (IQR) is shown as solid blue boxes, the 1.5-times IQR range as line with ending, and the outliers are marked as a +. The median is marked in orange.

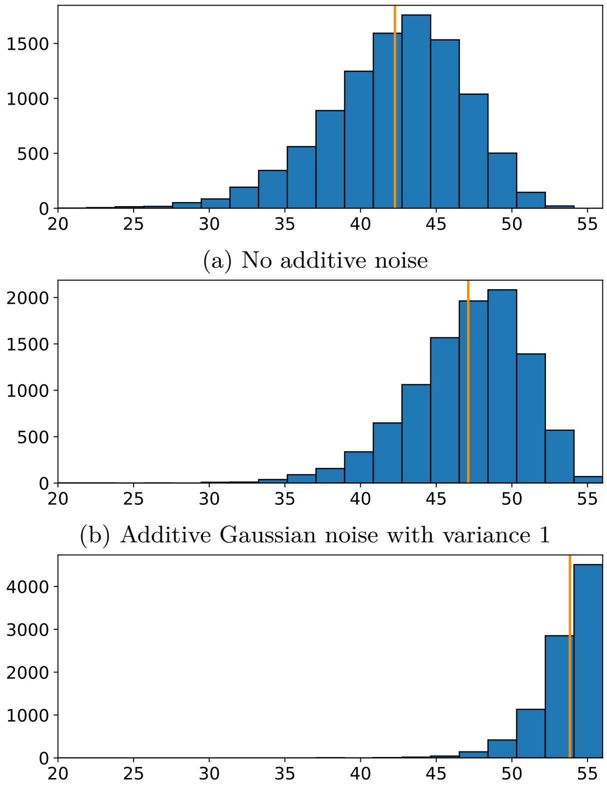
generality of this observation. The question is, whether weak keys are caused by the DES key-schedule or the hardware implementation. The simulation assumes perfect Hamming distance leakage of successive round keys, where the contribution of each bit in the registers is equal. The simulation takes DES keys as input, computes the 16 successive round keys and creates simulated side-channel traces by counting the sum of XOR differences between successive round keys at 15 updates. The artificial traces can be seen as abstract current consumption values.

The model deliberately disregards certain noise factors, like different kinds of electrical noise and switching noise from other sources than the key-schedule. For instance, the simulation does not consider the main data path of the DES computation, which would introduce uncorrelated noise from independently processed values. This leads to results which are systematically better than attacks on measured traces. The simulation is, however, realistic regarding the constant switching noise which is created by all parts of the key except for the respectively currently profiled subkey. This switching noise is due to the fact that a profiling of the entire key at once is impossible, and always present, even for repeated measurements and under perfect conditions since the key remains constant and the leakage of the key-schedule depends only on the key itself.

Compared to real devices, the model is simplified such that every bit position in the register leaks the same amount of information. This implies that any lower bounds for the security levels derived from simulations need to be applied with caution. We also added Gaussian noise with different variance (and mean value zero) to observe the dependence of results on the SNR. Note that compared to the actual device, the model does not include the backward computation of eight rounds because they do not introduce new information compared to the forward rounds, at least in the setting without additive noise.

Even for simulated traces without additive noise, the profiling set needs to have a sufficient size. Profiles for each subkey are created from traces with different keys but equal subkey value, such that the switching noise from the other key parts is averaged. As an example, simulating 1 million traces means that every class of the 7 bit XOR templates can be profiled using $10^6/2^7 \approx 78k$ traces. On the other hand, if the attacker is confronted with only measurements for a fixed key he cannot eliminate switching noise from other subkeys. Hence, without further additive Gaussian noise, one simulated trace per key already corresponds to the best case for an attacker in this scenario.

5.6.1 Distribution of security levels for random keys is also wide



(a) No additive noise

(b) Additive Gaussian noise with variance 1

(c) Additive Gaussian noise with variance 10

Figure 17: Histograms for security levels of 10k simulated traces, one attack trace per key. Security levels are plotted on the x-axis, number of keys per bin on the y-axis. The mean security level is marked by an orange line.

We simulated a profiling set and an attack set and performed an attack identical to the attacks on measured traces. We used 1 million random keys for profiling and an additional set of 10k random keys for single trace attacks. Figure 17a depicts the results without additive noise. The mean security level is 42.3 bit, while the minimum among the 10k randomly selected keys is 20.1 bit. As expected due to the described simplifications, this is much lower than the 49.4 bit average security level obtained for single trace attacks on measured traces in Table 1. Importantly, the security levels of the 10k different keys are widely distributed, similar to what is observed in the measurements. *This confirms that keys have inherently different security levels in a simple model, which leaks the Hamming distances of subsequent DES round keys.* These results are independent of other implementation and side-channel measurement details or any particular device.

A lower SNR naturally leads to higher average security levels. Figure 17b depicts the case of additive Gaussian noise with variance 1, the distribution has a mean of 47.1 bit. Shown in Figure 17c is the case of additive Gaussian noise with variance 10, where the average security level is 53.8 bit. The distributions observed in simulations with additive noise are similar to the results of actual attacks. So far, however, these results do not tell us how well the simulated security levels predict results obtained from measurements and more specifically, if weak keys in the simulation are weak on the device.

5.6.2 On different template lengths

To put our choice of templates in context, we tested different template lengths and various amounts of overlap between subkeys.

Table 2: Average security levels after attacking with different templates for 10k keys each.

	XOR templates		
subkey length for profiling [bit]	7	8	9
subkey overlap [bit]	0	1	0
subkey length for key rank [bit]	7	7	9
Mean [bit]	**42.3**	42.1	41.5

In Table 2, we present results for two additional cases. First, we used 8 bit instead of 7 bit templates for profiling, which results in 1 bit overlap between templates in order to fit four templates into the 28 bit of a register. Marginalization before the key rank estimation leads to the same subkeys as for 7 bit templates. Thus, the difference between the two is only the exploitation of 4 additional strong transitions during the attack phase. The benefit is small, as expected. Secondly, we investigated 9 bit templates, which perfectly cover the 27 independent XORs of each register with three subkeys and without overlapping bits. Therefore, the total number of subkeys is reduced to only 6 instead of 8 and the main benefit should be through a reduction in switching noise. Indeed, we gain roughly 1 bit in the average security level compared to 7 bit templates. As a downside, however, we would need two and four times as many traces for the profiling of measured traces with 8 and 9 bit templates, respectively.

5.6.3 Predicting security levels

Based on the previous results, we assumed that there is a correlation between security levels of keys after measurement on a DUT and results from simulations. In this section we show the accuracy and limitations of predicting security levels through simulations. We used the same set of keys for simulations and measurements and compared the results. We investigated two groups of keys:

1. 200 keys with key bits Bernoulli(p)-distributed, where the probability of a key bit being zero is either $p = 0.1$ or $p = 0.9$. The bits of these keys are therefore almost all ones or all zeros.

2. 350 keys selected evenly from the range of security levels out of 10k uniformly random keys. Thus key bits are distributed as Bernoulli(1/2).

While keys from the second group were generated from a uniform distribution as recommended for maximum security, the keys in the first group are unlikely to appear in practice. The skewed ratio of zeros and ones ($p \in \{0.1, 0.9\}$) for key bits in the first group leads to fewer XOR-transitions during the key-schedule. These keys are expected to have lower than average security levels and simpler leakage behavior.

For both classes, we performed attacks on noiseless simulated traces as well as on real measurements. In the latter case, we used 1000 measurements for each key to reduce noise factors to a minimum and make both attacks more comparable.

Figure 18 depicts the results for both classes as scatter plots on a x-y diagram with the simulation results on the x-axis, and the measurement results on the y-axis. Security levels for the first group ($p \in \{0.1, 0.9\}$) are shown in red, for the second group ($p = 1/2$) in blue. Linear approximations for both groups highlight the relation between measurement and simulation. The slope of the regression line for the first group is 1.03, for the second

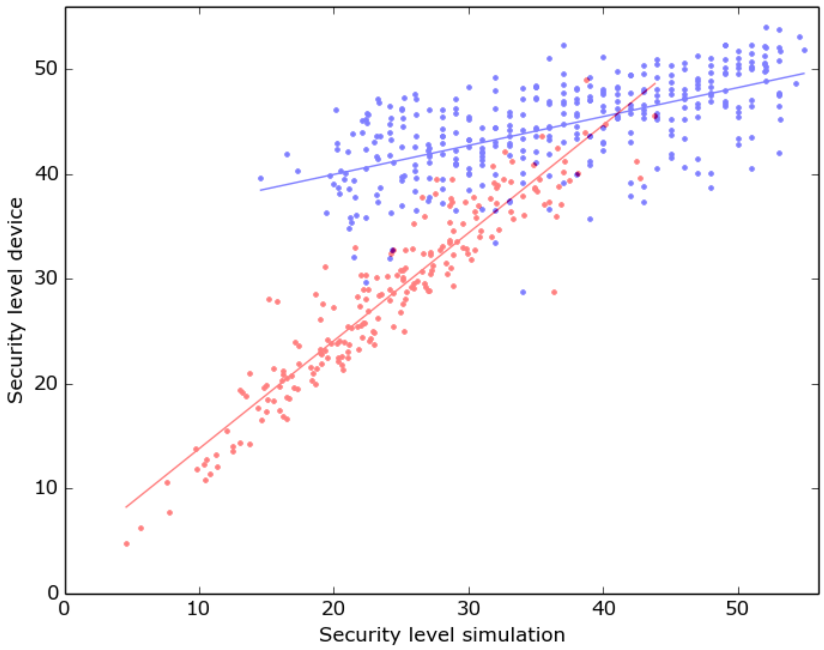


Figure 18: Two groups of random keys, one from a uniform distribution (blue) and one with skewed ratio of zeros and ones (red, either 90% zeros or 90% ones). Security levels after attacking simulated and measured traces are shown on the x-axis and y-axis, respectively. Linear approximations are plotted for both groups.

0.28. Both groups show a dependence between simulation and measurements on the actual device. This means that in addition to similar distributions of security levels, the simulated leakage actually allows some predictions for attacks on real devices. Both groups also show significant variation of security levels around their linear approximation, which cannot be attributed to noise in the measurements since 1000 traces per key were used in the attacks (cf. Sec. 5.5.2). The variations are likely caused by simplifications of the simulation.

However, the quality of the predictions is very different for the two groups of keys. The first group (red, $p \in \{0.1, 0.9\}$) shows a close to perfect match between simulation-based prediction and results achieved from actual measurements. A perfect match would be represented by a 45° line through the origin. In this group, weak keys seem to be weak independent of the device. On the other hand, the second group (blue, $p = 1/2$) shows a significantly lower prediction capability of the simulation. For keys in this group, the results may depend strongly on the device, although this was not tested.

A possible explanation for the difference of results between the two groups of keys is that the simulation model weights the leakage from every bit in the two registers equally. Keys in the first group (red, $p \in \{0.1, 0.9\}$) have fewer bit transitions in the registers during the key schedule, which limits the impact of the differences of contribution of register bits. Furthermore, the impact of switching noise is reduced as well, since most subkeys contribute little to none to the leakage. This could explain why the prediction is relatively accurate for those keys. Even within these keys, results range from security levels of ≈ 4 to ≈ 45 bit. This shows that constructing weak keys is in fact more complex than simply aiming for skewed ratio of zeros and ones in the key bits.

The prediction for the second group of keys (blue, $p = 1/2$) would suffer more from the simplification of the simulation, since more transitions and all subkeys are contributing to leakage. It would be possible to parameterize the simulation model and extract weight parameters for the register bits. This would likely allow an improved prediction also for uniformly random keys for this specific device and measurement setup. However, the model would lose generality at the same time.

5.6.4 Security levels significantly depend on switching noise

We have already seen that keys with a skewed ratio of ones and zeros in their key bits are weaker than average. This section further investigates why some subkey values are easier to classify than others by examining the generated templates of subkeys which are profiled based on simulated traces.

The success of a template attack depends on the ability to classify subkey values given the templates. Each template is determined by its mean trace and a pooled covariance matrix. If the mean traces of two classes are very similar, an attacker has a low chance of distinguishing them. If they are far apart relative to the respective covariances they can be distinguished easier. The chance of a correct classification can, thus, already be limited due to high similarities of profiles.

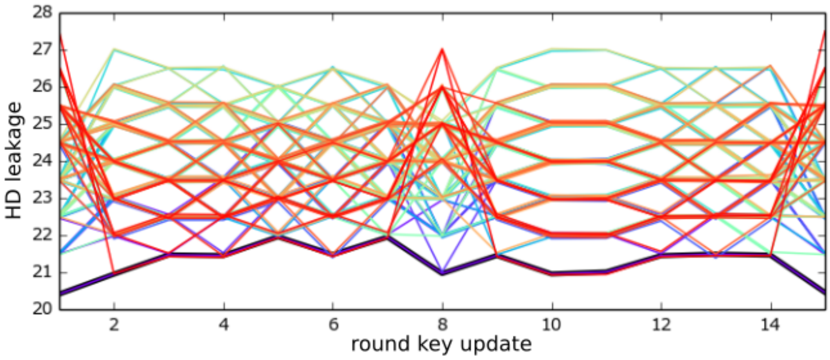


Figure 19: Simulated trace means for all 128 classes of one subkey using 1 million traces. The mean trace of the class with index 0 is highlighted in black.

In the following, we profile one individual subkey (as it is common). We chose the first out of the 8 available subkeys in this example (7 bit XOR templates). All other subkeys contribute switching noise. Figure 19 depicts the trace means of the 128 classes of this one subkey. The x-axis shows 15 time points where round-key transitions occur. The means are generated using 1 million simulated traces with random keys. The y-axis depicts the mean traces of all 128 classes computed from $\approx 10^6/128$ simulated traces each. For every specific value of the first subkey, different random keys (with different values in the other subkeys) are averaged. Each trace is computed by summing the XOR transitions in the two 24 bit round key registers for the 15 time points, which means the values before averaging range between 0 and 48, a part of which is the contribution of the targeted subkey, and the rest is contributed from the other subkeys as switching noise. They are then averaged according to their subkey values. In the figure we see the effect of templates averaging-out the switching noise from all keys parts other than the profiled subkey, such that the profiles only range between 20.5 and 27.5. There is significant variance in the traces (represented in the covariance matrix of the templates) due to the variations, i.e. switching noise, of these other subkeys.

Note that some classes stand out by attaining minimal or maximal values in some rounds. The class with index zero (highlighted in black in Figure 19) attains the minimum in all 15 rounds, representing the cases in which all subkeys are the same.

The observable difference of rounds 1,8 and 15 to all other rounds stems from the fact that at these rounds the shift operation of the key-schedule shifts by one bit rather than two. As described in Section 4.1 and depicted in Figure 6, some XORs occur often (e.g. transitions 2 to 7 and 9 to 14), others less frequently (transitions 1, 8, and 15). The existence of the two groups explains that the simulated data is also different for those two groups.

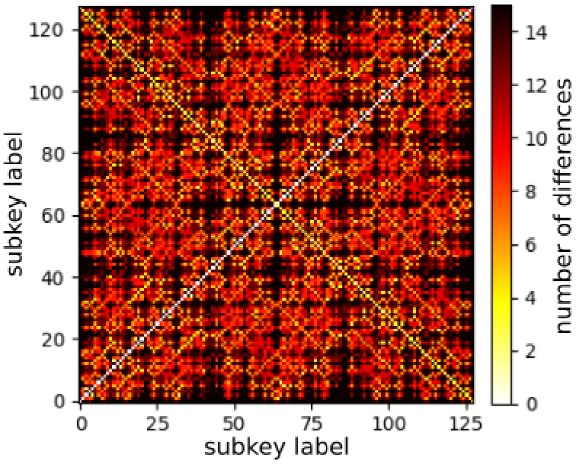


Figure 20: Number of differences between pairs of simulated trace means for all 128 classes of one subkey using 1 million traces. Pairs of identical profiles appear as white pixels.

Figure 20 visualizes the 'exploitable' *differences* between the profiles. For every pair of profiles from the set of 128 subkey values, we calculate the number of rounds in which they are different, using the fact that the remaining noise on the profiles is much smaller than 1. Any pair of profiles that are identical in every round appears as white pixel in the plot. *As important observation, we find that apart from the diagonal, there are no other pairs of identical profiles* (even if there are similarities and low differences). This holds true for all other subkeys of 7 bit XOR templates and for 10 bit XOR templates (data not shown). *We conclude that for XOR templates every subkey profile is in fact unique after switching noise is (de facto) removed. Thus, any misclassification during the matching phase of a template attack must be caused by switching noise.*

This does not fully answer the question, whether the variation of security levels is caused by inherent leakage differences or switching noise. One would have to either profile the entire key or try to estimate the entropy in the leakage. The first approach is clearly made impossible by computational limitations. The second approach is computationally more feasible and was attempted by Wagner et al. [WHG17]. However, as discussed in Section 2.4, they made an incorrect simplification to speed up calculation time. Still, their results seem to suggest that for the entire key the profiles are not unique, which is in contrast to our results on subkeys of XOR templates. Note that the difference between profiling bit values and XORs only leads to a constant offset of results by 2 bit and cannot explain the variation of security levels.

Finally and although we cannot fully answer the question as to how the variation of security levels arises, we can make a *clear statement for any practical attacks. In practice, the key needs to be profiled in parts of less than* ≈ 12 *bit at once, and in this case, the spread of the distribution is caused by switching noise alone.* All methods discussed in this paper can still be applied in more general scenarios, irrespective of the exact cause of the distribution in subkey rankings or security levels.

5.6.5 Lower bounds on the security level

It is not helpful to use the minimal security level of all keys as a general lower bound or to assess security of an implementation. Keys with key bits all zeros or all ones in each register are the weakest, as their profiles are the most extreme. However, those exact keys are also cryptographically weak and using them is not recommended. The simulation shows a security level of 2 bits for those keys, which is the minimum possible value given XOR leakage. Actual measurements on the device with 900 measurements per key lead to results of 2 bits for those keys as well. Hence, the lower bound of security levels (including cryptographically weak keys) for attacks against single DES in this device is 2 bit. This does not provide much insight into the security of a device.

In general, the distribution of security levels obtained through simulations could be considered a worst case scenario. Although, some caution needs to be applied, since choices in the model do not necessarily lead to minimal possible security levels. The leakage from each register bit is treated equally, while in reality their contribution to overall leakage varies. Also, the choice of bit length for the templates is arbitrary. By doubling the effort one can always use one bit more per template and thus reduce switching noise. However, we perform all simulations and practical attacks using 7 bit templates, which makes a comparison easier. A worst case for security levels independent of the actual attack could only be achieved by analyzing the entropy of traces directly, which is computationally difficult and has little practical relevance.

Finally, it is not even clear how to compare two distributions of security levels, as already discussed in Section 5.4. Therefore, simulations primarily provide lower bounds on the mean or different percentiles of the distribution.

6 Impact on triple-DES

So far, this contribution has described results of side-channel attacks against single DES. However, the last application of DES that provides reasonable security guarantees today is in the form of triple-DES. The impact of the described attack should therefore be measured by its implication for triple-DES implementations. Note that the use of any form of DES is generally not recommended by the German BSI [Bun19].

The best known cryptographic attacks against triple-DES are meet-in-the-middle attacks [DH77]. It is reasonable to adapt them to deal with side-channel results. This is challenging, because largely varying security levels (distributions as observed above) need to be handled. The output of a key rank algorithm, i.e. enumerated lists of key candidates including associated probabilities, must be used in an optimal way. In the following, we present a meet-in-the-middle attack on SCA results with *wide distributions* of security levels. We calculate the distribution of security levels for a 3-key triple-DES under such attack results from single DES. This allows us to estimate the percentage of key-triples that are threatened by successful brute-force attacks after side-channel analysis. For example, this will allow to estimate the percentage of keys going below 80 bit brute-force effort after a side-channel attack.

6.1 Recap on the cryptanalytic security of triple-DES

The U.S. NIST seems to still accept the use of 3-key triple-DES [Nat16]. The cryptanalytic security level of a 3-key triple-DES is equal to twice the key length due to meet-in-the-middle attacks given three known-plaintext and ciphertext pairs [DH77, MH81] (one pair for the main attack algorithm, two more to rule out wrong candidates).[4]

A meet-in-the-middle attack on 3-key triple-DES, which encrypts a plaintext P as $C = \mathrm{DES}_{k_3}(\mathrm{DES}^{-1}_{k_2}(\mathrm{DES}_{k_1}(P)))$, works as follows [vOW90]. First, a known plaintext is used to exhaustively create and store a list of the first intermediate value $a_i = \mathrm{DES}_{k_{1(i)}}(P)$ for all 2^{56} possible candidates of $k_{1(i)}$. This requires $\mathcal{O}(2^{56})$ computations and $\mathcal{O}(2^{56})$

[4] Access to three pairs of plain- and ciphertext seems reasonable for many applications.

storage. Then, the ciphertext C is used to exhaustively test (k_2, k_3) by backwards-computing $a' = \mathrm{DES}_{k_2}(\mathrm{DES}_{k_3}^{-1}(C))$. This requires $\mathcal{O}(2^{112})$ computation. Every result a' is compared to the list a_i. If a match $a' = a_i$ is found, the respective keys k_1, k_2, k_3 are candidates for the correct keys. Two more plain- ciphertext pairs are used to eliminate false positives[5]. In summary, this algorithm requires $\mathcal{O}(2^{112}) + \mathcal{O}(2^{56}) = \mathcal{O}(2^{112})$ computation and $\mathcal{O}(2^{56})$ storage. Note that the expected security level of a cryptanalytic meet-in-the-middle attack has an expected or average complexity of $\mathcal{O}(2^{111})$.

6.1.1 Two key triple-DES

Although neither the NIST nor the BSI approve it, 2-key triple-DES is still in use in some applications. The van Oorschot-Wiener attack [vOW90] is an effective known-plaintext attack against 2-key triple-DES, with a complexity of $\mathcal{O}(2^{121-n})$ for 2^n plain- and ciphertext pairs. It reduces the security to $\approx$ 80 bit, given 2^{32} plain- and ciphertext pairs. Here, we are mainly interested in single-trace attacks, which makes this attack less relevant. The van Oorschot-Wiener attack is more efficient than the meet-in-the-middle attack only if a sufficient number of plain-/ ciphertexts is available. Further, side-channel information of the key-schedule has little use for the van Oorschot-Wiener attack, as two tables of size $\mathcal{O}(2^{56})$ must be created and only one of them would benefit from key rankings. The complexity of creating tables reduces at best from $\mathcal{O}(2^{57})$ to $\mathcal{O}(2^{56})$ and overall would still be at least $\mathcal{O}(2^{120-n})$.

When using the meet-in-the-middle attack on 2-key triple-DES, one saves the effort of creating the list $\{a_i\}$, removing all memory requirements from the algorithm. Further, SCA gives slightly better estimates for the key part $k_1 = k_3$, whose leakage appears twice in each trace. Apart from this, the following discussion generalizes to 2-key triple-DES, if few plain-/ ciphertext pairs are available.

6.2 Side-channel security of 3-key triple-DES allowing a meet-in-the-middle advantage

In the following, we discuss the overall security of 3-key triple-DES based on the side-channel security of a single DES by using a modification of the regular meet-in-the-middle attack. The side-channel security of a 3-key triple-DES depends on the results of three simultaneous attacks on three different keys. While the maximal complexity of a meet-in-the-middle attack is still $\mathcal{O}(2^{112})$, we should expect a significant reduction of the expected security level by using information from the attack.

The side-channel attack leads to a different entropy reduction for every one of the three keys k_i, $i \in \{1, 2, 3\}$. However, the attacker can a-priori not know, which of the three keys is better than the others, since the security levels of the keys are independent and identically distributed random variables. Consequently, referring to the description of the meet-in-the-middle attack above, it does not matter whether we choose to create the list using k_1 and match it after the first encryption (and treat k_2 and k_3 together as one 112-bit key) or before the last encryption. Both options are equally likely to succeed faster than the other and we may therefore pick one without loss of generality.

First, we need to combine our key rank results for two of our keys, say k_2 and k_3, to obtain a key ranking for the combined key-pair (k_2, k_3). This can be done by applying a key enumeration algorithm to the candidate lists of both keys (including associated probabilities), or by treating the pair (k_2, k_3) as one 112-bit key already during the attack. Both options are equivalent.

For the actual side-channel meet-in-the-middle attack, we want to process the key candidates of k_1 and $k_{2,3} = (k_2, k_3)$ in order of their likelihoods. That is, we want to

[5] False positives are possible because the input and output lengths are shorter than the key length.

successively check all candidates up to ranks r_1 and $r_{2,3}$ in both lists such that we maximize the probability

$$\mathsf{P}\left(R(k_1) \leq r_1 \text{ and } R(k_{2,3}) \leq r_{2,3}\right) = \mathsf{P}\left(R(k_1) \leq r_1\right) \cdot \mathsf{P}\left(R(k_{2,3}) \leq r_{2,3}\right) \qquad (8)$$

in each step, where $R(k)$ denotes the rank of a key. The cumulative probabilities on the right-hand-side of the equation are a direct output of the key enumeration algorithm applied to the two key parts k_1 and $k_{2,3}$. We then create two lists, $\{a_i\} = \{\mathrm{DES}_{k_{1(i)}}(P)\}$ and $\{a'_i\} = \{\mathrm{DES}_{k_2}(\mathrm{DES}^{-1}_{k_3}(C))\}$, up to indices r_1 and $r_{2,3}$ respectively. In practice, we would start at the top of both lists (assuming candidate lists are obtained by key enumeration and thus ordered by likelihood) with $r_1 = 1$ and $r_{2,3} = 1$. We then add a candidate to one of the lists $r_1 = r_1 + 1$ or $r_{2,3} = r_{2,3} + 1$, according to which new candidate has the higher likelihood by Equation (8). Each time, we add a candidate to one of the lists, we check for a new match $a_i = a'_j$ for some $i \in [1, r_1]$ and $j \in [1, r_{2,3}]$. Any match can either be confirmed or disregarded by testing with the two additional plain- and ciphertexts until we succeed to find the correct key-triple (k_1, k_2, k_3). This algorithm does require the same amount of storage as number of DES computations (which is more storage than required by the original algorithm for levels > 56). If storage is an issue and in particular if one is not interested in very low security levels, i.e. < 56 bit, it is more practical to use the following simplified algorithm instead.

Simplifying the algorithm to estimate security levels > 56 bit. The above described algorithm is relatively complex and requires more storage than the original meet-in-the-middle algorithm. Fortunately, we may simplify the side-channel meet-in-the-middle algorithm for cases > 56 bit expected security level by using the fact that one of the lists, $\{a_i\} \in \mathcal{O}(2^{56})$, is much shorter than the other, i.e. $\{a'_i\} \in \mathcal{O}(2^{112})$. This helps because it is acceptable to loose the ability to precisely determine the security level ≤ 56 nowadays where security levels of > 80 are especially relevant. For the simplified algorithm, we first calculate the complete list $\{a_i\} = \{\mathrm{DES}_{k_{1(i)}}(P)\}$, which has memory and computational complexity of $\mathcal{O}(2^{56})$. We then iterate over the second list $\{a'_i\} = \{\mathrm{DES}_{k_2}(\mathrm{DES}^{-1}_{k_3}(C))\}$ in order of their probabilities and check each candidate for a match with $\{a_i\}$, which has no additional cost on memory. The overall expected computational complexity of the algorithm is $\mathcal{O}\left(\max\left\{2^{56}, \mathsf{E}[R(k_{2,3})]\right\}\right)$.

Use cases of the simplified versus exact algorithm. Since in all cases studied by us the probability $\mathsf{P}\left(R(k_{2,3}) \leq 2^{56}\right)$ is very small, the attacker is almost always at an advantage when picking the simplified algorithm. There is no penalty in computational cost and the memory requirements are fixed and almost certainly below those used in the exact algorithm. This advice might be specific to our case and depends very much on the observed distributions. Even when using the exact algorithm, the following estimates on computational complexity based on the simplified algorithm still provide an upper bound.

Extrapolating triple-DES estimates from single DES results. Under the assumption that all three DES-executions in a triple-DES lead to identical distributions F_S^{single} of security levels, we may use the corresponding density f_S^{single} obtained from attacks on single DES to extrapolate to triple-DES. Unfortunately, we do not have a closed form expression for the density of the rank or security level, so they have to be estimated directly through histograms from empirical results. Any numerical calculations on distributions should be done for security levels instead of key ranks, to make them numerically stable and benefit from better density estimates on security levels. We first need the distribution of the combined security level of two keys $F_S^{2-\text{key}}$, which is

$$\begin{aligned}F_S^{2-\text{key}}(s) &= \mathsf{P}(S(k_2) + S(k_3) \leq s)\\ &= \int_0^{56} \mathsf{P}\left(S(k_2) \leq s - x \middle| S(k_3) = x\right) \mathsf{P}\left(S(k_3) = x\right)\, \mathrm{d}x\\ &= \int_0^{56} F_S^{\text{single}}(s-x) f_S^{\text{single}}(x)\, \mathrm{d}x\end{aligned}$$

For the density we get

$$f_S^{2-\text{key}}(s) = \int_0^{56} f_S^{\text{single}}(s-x) f_S^{\text{single}}(x)\, \mathrm{d}x \tag{9}$$

accordingly, which is the convolution of the two single-key densities. Note that $f_S^{\text{single}}(x) = 0$ for any $x < 0$ or $x > 56$.

Further, since $\mathsf{P}\left(S(k_{2,3}) \leq 56\right)$ is very small in practice, we also get a direct approximation of the distribution and expectation of security levels for three keys. We can express the expected security level for triple-DES after a meet-in-the-middle attack using the empirical density estimate in Equation (9) as

$$\mathsf{E}\left[S(k_1, k_2, k_3)\right] \approx \mathsf{E}\left[S(k_{2,3})\right] = \int_0^{112} s f_S^{2-\text{key}}(s)\, \mathrm{d}s,$$

while the expected computational complexity of the simplified algorithm follows from

$$\mathsf{E}[R(k_{2,3})] = \int_0^{112} 2^s f_S^{2-\text{key}}(s)\, \mathrm{d}s.$$

6.3 Triple-DES estimates based on measurement data

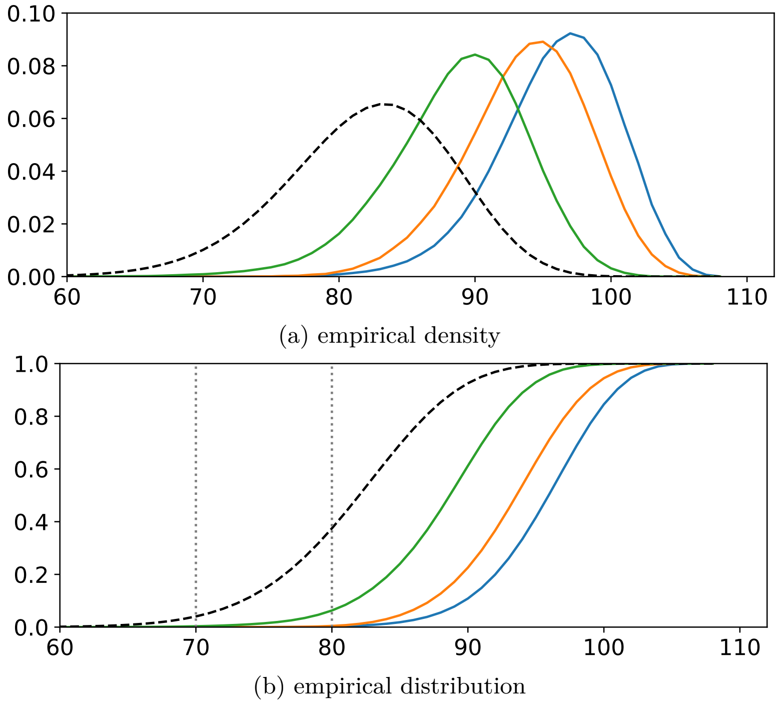
For the estimation of triple-DES security levels of an actual device, the distribution of a single DES is required. It can be obtained from actual measurements, in our case after attacking 1, or 3, or 900 traces per key. Every case represents how many measurements the attacker would be able to perform. We computed results for those three cases of 1, 3, and 900 traces per key using the results presented in Figure 14 and Table 1.

Figure 21a depicts the empirical densities of the security level distributions after 1 (blue), 3 (orange), and 900 (green) traces per key when attacking 3-key triple-DES. It also includes results obtained from a noise-free simulation (black, dashed). This, as described above, precisely represents the security level of the 3-key triple-DES for all cases with a security level > 56 bit. Figure 21b depicts the corresponding empirical cumulative distributions.

Table 3: Security levels for 3-key triple-DES based on measured single DES results, and simulation without additive noise.

	1 trace	3 traces	900 traces	sim.
Mean sec. level 1-key 1-DES [bit]	49.4	48.2	45.7	42.3
Mean sec. level 3-key 3-DES [bit]	96.1	93.8	88.7	82.1
Fraction of 3-key 3-DES cases < 80 bit	0.24 %	0.43 %	6.3 %	37.4 %
Fraction of 3-key 3-DES cases < 70 bit	0.0015 %	n.a.	0.32 %	4.0 %

Table 3 summarizes the concrete values for some exemplary cases. First, the mean security level for 3-key triple-DES for the analyzed device for actual single trace attacks is



(a) empirical density

(b) empirical distribution

Figure 21: Empirical density and distribution of security levels for an attack against 3-key triple DES (equal to combined attack on two DES keys). Results are shown for attacks on 1 trace (blue), 3 traces (orange) and 900 traces (green). Results obtained from a noise-free simulation are shown (black, dashed).

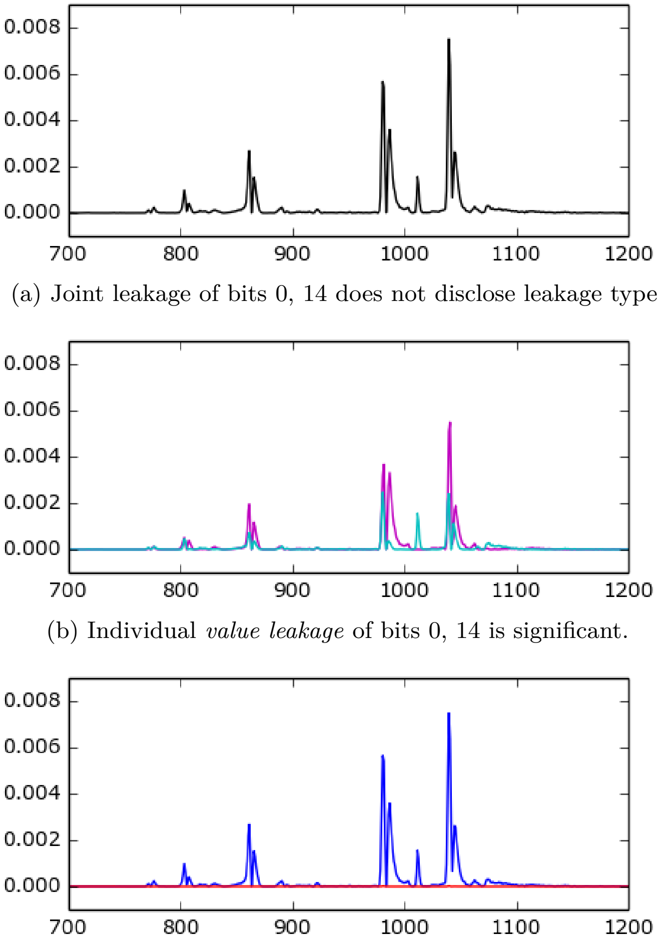
96.1 bits. *As one of the most important observations, from all possible key-triples, only a fraction of* 0.24 % *will lead to a 3-key triple DES security level of* $<$ 80 *bits in a single trace attack.* An attacker can increase this percentage to 0.43 % if he is allowed to perform 3 measurements, and to 6.3 % if he is able to increase the number of measurements to a large number of traces, e.g. 900. Note, that no further improvement can be expected since security levels reach their minimums already at about 200-400 traces. *However, even with 900 traces per key, the attacker can achieve a security level of* $<$ 70 *bits only for* 0.32 % *cases. This means that only every 300-th device will exhibit a security level* $<$ 70 *bits when a large number of measurements per key is allowed.*

The results for a noiseless simulation are given as a reference. They cannot be regarded as a true lower bound, however, since the model is simplified. This simplification may in part be unfavorable for the attacker (e.g. differently weighted transitions may be easier to distinguish for the attacker).

7 Empirical study: General purpose microcontroller

We repeated the described attack on the DES hardware accelerator of an STM32F4 general purpose microcontroller. The device is studied after decapsulation from the top side of the die using a 250 µm diameter EM probe and a sampling rate of 2.5 GS/s. The measurement position and 600 POIs for the template attack were chosen based on a CPOI leakage test.

As emphasized before, it is crucial to understand the exhibited leakage of the device to make the right choices for the model in the template attack (see descriptions in Section A.4 and Section 3). We performed a test to determine the leakage type similar to Section 5. The results are discussed here for key bits 0 and 14. Figure 22 shows the computed SNR values,



(a) Joint leakage of bits 0, 14 does not disclose leakage type.

(b) Individual *value leakage* of bits 0, 14 is significant.

(c) No XOR leakage (red), only independent leakage of bits (blue)

Figure 22: SNR (y-axis) for the bit pair 0,14 during the time period, where key-dependent leakage was observed (x-axis, samples recorded at 2.5 GS/s). The joint leakage can be explained entirely by the *value leakage* of the bits. There is no XOR leakage.

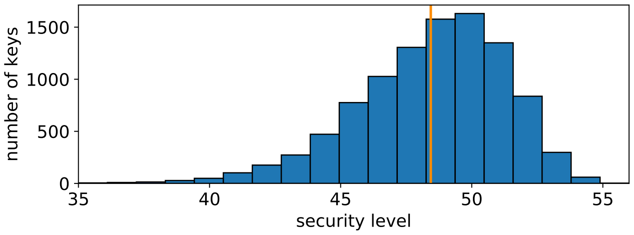


Figure 23: Histogram of security levels after template attacks on 10k keys using 100 traces per key on STM32 hardware DES accelerator. The average security level, marked by an orange line, is at 48.4 bit.

starting with the joint SNR of both bits in black in the upper graph. There is exploitable leakage, which could be caused by any leakage model. From the middle graph, it can be seen that significant value leakage of the bits is present and exploitable as well. The lower graph shows that the entire joint leakage can be explained by the sum of independent value leakages of the two bits, depicted in blue. There is virtually zero additional XOR leakage, which is shown in red. Identical results were observed for other tested bit pairs (data not shown). *We conclude that the device exhibits value leakage from individual key bits, but no XOR leakage.* Consequently, this device has a completely different leakage model than the one investigated in Section 5, which has pure XOR leakage.

According to Section 3.1.1, this means that key bit *values* (not their XORs) should be targeted directly during profiling, and that key enumeration can be performed in a straight-forward manner (not on XORs) due to the independence of individual bits. We chose 7 bit templates for classifying key bit *values* during the attack. Therefore, the model used for the template attack is also different to the case described in Section 5 where XOR leakage is targeted, and enumeration performed on XOR values.

Figure 23 shows the resulting security levels after attacking 10k keys using 100 traces per key. We can observe a *wide distribution* of security levels again, with a mean security level of 48.4 bit. This is similar to the empirical study in Section 5, although device, implementation and leakage model differ between the two empirical cases. Specifically, the previously observed XOR leakage between round key bits is missing here. Nonetheless, the *distribution* of security levels is similar, including the observation of *weak keys.* These results support the conclusion that any key-schedule implementation with non-uniform distribution of leakage may have varying security levels after a template attack. In particular, this case study proves the point for Hamming weight leakage of a DES key-schedule.

8 Empirical study: Second Security Controller

We repeated the described attack against a device of another family of security controllers. The measurement setup and processing steps are very similar to Section 5. A number of 460 POIs was chosen for the template attack. Figure 24 shows the resulting security levels after attacking 1k keys using 900 traces per key. We can observe a *wide distribution of security levels again*, with a mean security level of 48.7 bit.

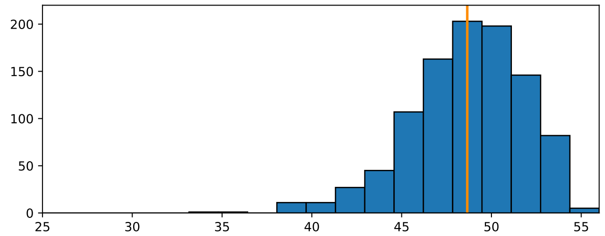


Figure 24: Histogram of security levels after template attacks on 1k keys using 900 traces per key. The average security level, marked by an orange line, is at 48.7 bit.

The results are comparable to the empirical study in Section 5 where the average security after using 900 traces is at 45.7 bit. The security level distribution is shifted to the right by ≈ 3 bit which means that the percentage of keys with lower security levels is *lower*. Consequently the number of keys with low security levels (e.g. below 80 bits) in a triple-DES use will also be *lower* while the general observation of keys with low security remains.

Additionally and similar to Section 5.6.3, we tested two different classes of keys in regard to their relation between simulated security levels and outcome of actual attacks. One class containing 288 random keys, the other containing 212 keys with key bits Bernoulli(p)-distributed, where the probability of a key bit being zero is either $p = 0.1$ or $p = 0.9$. The bits of these keys are therefore almost all ones or all zeros. For both classes, we

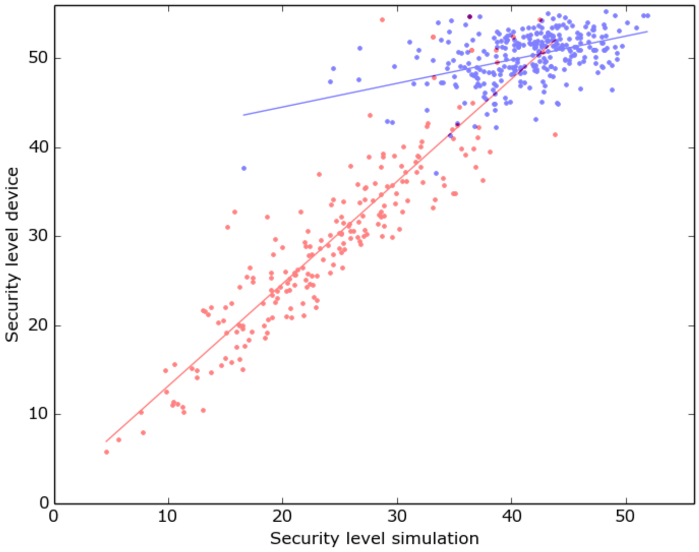


Figure 25: Two groups of random keys, one from a uniform distribution (blue) and one with skewed ratio of zeros and ones (red, either 90% zeros or 90% ones). Security levels after attacking simulated and measured traces are shown on the x-axis and y-axis, respectively. Linear approximations are plotted for both groups.

performed attacks on noiseless simulated traces as well as on real measurements using 700 measurements. Figure 25 depicts the results for both classes in a similar manner as in Section 5.6.3. The slope of the red regression line is 1.15, for the blue one 0.27. The results are comparable. The same strong dependency between simulation and actual attacks is detectable for keys with extreme Hamming weights.

9 Conclusion

The investigation answered many important questions regarding profiled side-channel attacks against implementations of the DES key schedule in general, and two commercial security controller products in specific. The main insight is that *weak* keys do exist after side-channel analysis of the DES key schedule, and that security levels are widely distributed. To judge implications for triple-DES we explain how 3-key triple-DES security levels can be estimated based on evaluating single-DES.

We report the following numbers for the investigated commercial security controller (first of two; results for the second security controller lead to slightly higher security levels). The mean security level for 3-key triple-DES on this device after single trace attacks is still relatively high with 96.06 bit. However, a fraction of 0.24 % key-triples exhibit security levels of < 80 bits after a single trace attack. If the attacker uses 900 traces instead of a single one per key, the fraction of key-triples < 80 bits increases to 6.3 %. The fraction of key-triples < 70 bits is 0.32 %. The resulting security levels are widely distributed and dependent on the key-value with tails reaching down to almost zero. A worst case of 2 bit of remaining security level for two specific DES keys, i.e. all zeros/ones is observed. (Note, however, that those keys are also cryptographically weak and unlikely.) Nonetheless, this indicates that basing the security estimates for a device on lower bounds or worst case results seems unreasonable. Assessing the device security based on the average security level also fails to acknowledge the full attack potential. Hence, the important question is: *How should the security level of a device after side-channel evaluation be assessed when results are widely distributed and weak keys are existent?* It seems like a security assessment in such cases needs to tolerate a certain non-zero, but low percentage of keys below a reasonable security level, such as e.g. 80 bits.

Our observation of widely distributed results and weak keys for profiled attacks against the DES key schedule seems independent of a specific device in the sense that (1) at least two frequently encountered leakage models of round keys are affected, and (2) that even a very simplified simulation model yields comparable results. From our analysis we conclude that *weak* keys are generally expected on all devices, while concrete results are of course in part device-specific. From this, it seems reasonable to assume that many more implementations may be affected *if* their DES key schedule is not adequately protected (i.e. through proper masking). Properties of the DES key schedule, i.e. mainly its linearity, seem to reinforce the issue which is yet another indication that DES is outdated.

This work raises the question whether such wide distributions of results and *the existence of weak keys* must also be expected in profiled side-channel attacks against the AES key schedule. While the AES key schedule includes non-linearity, a profiled attack against intermediate variables of the key schedule should likely lead to situations where some values are classified better than others. The AES key-schedule's non-linearity only means that such effects will likely only affect few or single round keys at once. In our opinion it seems likely that all profiled attacks against leaking key schedules will lead to similar key-value-dependent outcomes[6]. The DES key schedule only seems especially prone to such weakness because of its linearity.

References

[BGH+15] Nicolas Bruneau, Sylvain Guilley, Annelie Heuser, Damien Marion, and Olivier Rioul. Less is more. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 22–41, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[Bun19] Bundesamt für Sicherheit in der Informationstechnik. BSI - Technische Richtlinie. Kryptographische Verfahren: Empfehlungen und Schlüssellängen. TR-02102-1, January 2019.

[CK13] Omar Choudary and Markus G Kuhn. Efficient template attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 253–270. Springer, 2013.

[DH77] Whitfield Diffie and Martin E. Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *IEEE Computer*, 10(6):74–84, 1977.

[DS16] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 240–262. Springer, 2016.

[GGJR+11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, 2011.

[6] This is different from the popular DPA attack scenario where key-dependent intermediate values are targeted which are mixed with multiple different input values. This makes value-dependence unlikely.

[GGP$^+$15] Cezary Glowacz, Vincent Grosso, Romain Poussier, Joachim Schüth, and François-Xavier Standaert. Simpler and more efficient rank estimation for side-channel security assessment. In *Fast Software Encryption*, pages 117–129. Springer Berlin Heidelberg, 2015.

[GS14] Vincent Grosso and François-Xavier Standaert. ASCA, SASCA and DPA with enumeration: Which one beats the other and when? In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 291–312. Springer, 2014.

[HMH$^+$12] Johann Heyszl, Dominik Merli, Benedikt Heinz, Fabrizio De Santis, and Georg Sigl. Strengths and limitations of high-resolution electromagnetic field measurements for side-channel analysis. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012.

[HZZW16] Yongbo Hu, Chen Zhang, Yeyang Zheng, and Mathias Wagner. Ciphertext and plaintext leakage reveals the entire TDES key. Cryptology ePrint Archive, Report 2016/1143, 2016.

[KB07] Boris Köpf and David A. Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 286–296, 2007.

[LWWW17] Yang Li, Shuang Wang, Zhibin Wang, and Jian Wang. A strict key enumeration algorithm for dependent score lists of side-channel attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 51–69. Springer, 2017.

[MH81] Ralph C. Merkle and Martin E. Hellman. On the security of multiple encryption. *Commun. ACM*, 24(7):465–467, July 1981.

[MM18] Daniel P. Martin and Marco Martinoli. A note on key rank. *IACR Cryptology ePrint Archive*, 2018:614, 2018.

[MMO18] Daniel P. Martin, Luke Mather, and Elisabeth Oswald. Two sides of the same coin: Counting and enumerating keys post side-channel attacks revisited. *IACR Cryptology ePrint Archive*, 2018:19, 2018.

[MOOS15] Daniel P. Martin, Jonathan F. O'Connell, Elisabeth Oswald, and Martijn Stam. Counting keys in parallel after a side channel attack. In *Proceedings, Part II, of the 21st International Conference on Advances in Cryptology — ASIACRYPT 2015 - Volume 9453*, pages 313–337, Berlin, Heidelberg, 2015. Springer-Verlag.

[MOP08] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[Nat16] National Institute of Standards and Technology. NIST special publication 800-57 Part 1 Revision 4. Recommendation for key management. Part 1: General, January 2016.

[PMMOS16] Daniel P. Martin, Luke Mather, Elisabeth Oswald, and Martijn Stam. Characterisation and estimation of the key rank distribution in the context of side channel evaluations. pages 548–572, 12 2016.

[SC04] S. Salvadore and P. Chan. FastDTW: Toward accurate dynamic time warping in linear time and space. In *3rd Workshop on Mining Temporal and Sequential Data*, 2004.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 495–513, 2015.

[SMY09] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 443–461. Springer, 2009.

[UHDS+18] Florian Unterstein, Johann Heyszl, Fabrizio De Santis, Robert Specht, and Georg Sigl. High-resolution EM attacks against leakage-resilient PRFs explained. In *Cryptographers' Track at the RSA Conference*, pages 413–434. Springer, 2018.

[UHDSS17] Florian Unterstein, Johann Heyszl, Fabrizio De Santis, and Robert Specht. Dissecting leakage resilient PRFs with multivariate localized EM attacks. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 34–49. Springer, 2017.

[UKM+18] Thomas Unterluggauer, Thomas Korak, Stefan Mangard, Robert Schilling, Luca Benini, Frank K. Gürkaynak, and Michael Muehlberghuber. *Leakage Bounds for Gaussian Side Channels*. Springer, 2018.

[VCGRS12] Nicolas Veyrat-Charvillon, Benoît Gérard, Mathieu Renauld, and François-Xavier Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *International Conference on Selected Areas in Cryptography*, pages 390–406. Springer, 2012.

[VCGS13] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Security evaluations beyond computing power. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 126–141. Springer, 2013.

[VCGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 282–296. Springer, 2014.

[vOW90] Paul C. van Oorschot and Michael J. Wiener. A known plaintext attack on two-key triple encryption. In *Advances in Cryptology - EUROCRYPT '90, Workshop on the Theory and Application of of Cryptographic Techniques, Aarhus, Denmark, May 21-24, 1990, Proceedings*, pages 318–325, 1990.

[WH17] Mathias Wagner and Stefan Heyse. Single-trace template attack on the DES round keys of a recent smart card. *IACR Cryptology ePrint Archive*, 2017:57, 2017.

[WH18] Mathias Wagner and Stefan Heyse. Improved brute-force search strategies for single-trace and few-traces template attacks on the DES round keys. Cryptology ePrint Archive, Report 2018/937, 2018.

[WHG17] Mathias Wagner, Stefan Heyse, and Charles Guillemet. Brute-force search strategies for single-trace and few - traces template attacks on the DES round keys of a recent smart card. *IACR Cryptology ePrint Archive*, 2017:614, 2017.

[WHZZ16] Mathias Wagner, Yongbo Hu, Chen Zhang, and Yeyang Zheng. Comparative study of various approximations to the covariance matrix in template attacks. *IACR Cryptology ePrint Archive*, 2016:1155, 2016.

A Background on tools, analysis, and measurements

In this Section, we describe all employed trace processing and analysis methods which are taken from the well-established state of the art. Where necessary, we describe our implementation choices on top of the respective publications. We also describe our high-precision EM measurement setup.

A.1 Alignment

Alignment refers to a post-processing step on recorded traces. Most side-channel analyses require alignment of traces in order to remove the effects of deliberate and uncontrollable misalignment sources (e.g. internal oscillators, random dummy cycles, random delays, unsynchronized operating system tasks). Such misalignment cannot be prevented during oscilloscope recordings, even by using multi-stage triggers and triggering on IO and power trace patterns.

Static alignment methods align a trace at one particular time sample or during a short time-span. Significant peaks in EM traces or edges in power traces can often be used for a first step of alignment. Sometimes, if a significant pattern has been identified, a simple pattern matching (least square matching) using a cut example trace can be employed. It then depends on the reasons for misalignment, how many samples before and after this point are aligned correctly. This may be sufficient if only a short time period needs to be analyzed. For example, if the pattern or timing of the targeted algorithm execution has been identified beforehand, then fewer samples need to be aligned correctly. In many cases, multiple alignment methods are cascaded.

If static alignment methods are not sufficient to align the required time-span due to strong jitter or random delays, dynamic alignment methods can be used. The disadvantage of dynamic alignment methods are increased runtime and reduced quality of local alignment. The most popular form of dynamic alignment is dynamic time warping. It is used to align all traces to a chosen reference trace, by locally minimizing a suitable distance measure, for example the mean absolute error or mean square error. A fast algorithm for dynamic time warping has been described by Salvadore and Chan [SC04] and is available as a module for python[7]. The FastDTW algorithm requires a radius parameter. A larger radius parameter forces the algorithm to search a larger range for the best fit. It can be difficult to visually determine the quality of this form of alignment. A sure sign that the radius parameter is too small is a large number of jumps in the time-warping function, which appear as constant parts in the aligned traces. The runtime increases significantly with larger radius parameters. From our experience with side-channel traces, we find that the required radius parameter mainly scales with the length of the traces and less with the amount of jitter. In order to work with small radius parameters ($r <= 20$), we therefore split our traces in short sections (each ∼5000 samples long) and align these against sections of the reference trace. The trace sections are chosen such that they are overlapping by at least twice the maximal jitter. The overlap can be discarded after aligning the pieces to the reference trace. The aligned sections (without the overlapping parts) can be concatenated to recover

[7] https://pypi.org/project/fastdtw/

a complete aligned trace. Due to this piece-wise alignment, small errors can appear at the edges of the sections, especially if the overlap is not large enough. In our experience, this disadvantage is greatly outweighed by the speed-up compared to larger radius parameters required otherwise.

A.2 Assessing the amount of side-channel leakage

In side-channel analysis, a selection of relevant time-samples in a trace, called points of interest (POIs) is usually performed as a first step to reduce the amount of data from traces. When using a high-precision EM measurement setup, the position of the probe also has to be optimized. For both goals, it is important to assess the amount of side-channel leakage that can be exploited for each targeted variable. This variable usually depends on a part of the secret key.

In most cases there are several different variables to be targeted for different parts of the key, which we refer to as subkeys. The optimal measurement position could be different for every part. Nevertheless, the measurement position is usually determined for one exemplary subkey or using a leakage test which targets the entire key. Unterstein et al. [UHDSS17, UHDS$^+$18] determine multiple measurement positions for different subkeys. They are in a setting, where an attacker is allowed unlimited measurements of the attacked key. Hence, they look for different measurement positions to target different subkeys optimally. In contrast, in our contribution the focus is on attacks with a limited number of observations per key. Hence, an attacker is unable to repeat the observation at different measurement positions, even if different positions would be better to attack different key parts. Nonetheless, profiling of leakage using an open device with an unlimited number of measurements is allowed.

Some of the following methods also apply to settings where the key is inaccessible and unknown to the attacker (e.g. correlation-based leakage test). Measurements are always multivariate with many time-samples. A leakage assessment is usually performed for every measurement position, and for every time-sample in an univariate manner when searching for optimal measurement positions and exploitable time-samples. Established univariate methods compute signal-to-noise ratios (SNRs) or correlation coefficients to measure and compare exploitable leakage. Sections A.2.2 and A.2.3 describe the most established approaches based on t-test statistics and the Pearson correlation coefficient respectively. Actual SNR-computations are generally more expensive. Hence SNR-based leakage tests are used less frequently.

In general, a leakage test is performed early during an analysis when only little information about the implementation is available. Still, such methods already make some assumptions about the leakage behavior of the implementation. They often assume that variables can be described sufficiently by multivariate Gaussian distributions, and it is often assumed that variables leak information in some specific form, for instance as their Hamming weight. Based on these assumptions, one performs a leakage test on the chosen variables. If no leakage is detected in this manner, other models are tested. As an example, the transition between two consecutive values in a storage cell would leak their Hamming distance (see Section 3.1). Such choices depend on the assumptions about the targeted algorithm and implementation.

A.2.1 Univariate SNR-based leakage test

One straightforward approach to leakage assessment is to compute a univariate SNR for the variables to be classified through their statistical moments. This is usually achieved by assuming the signal can be characterized through the mean value of traces in a class exhibiting the same value and weighing the distance between class means by their inter-class variance. The SNR of the exploitable signal of a variable in side-channel data was established by Mangard et al. [MOP08] as

$$\mathrm{SNR} = \frac{\mathsf{Var}(\mathrm{Signal})}{\mathsf{Var}(\mathrm{Noise})} = \frac{\mathsf{Var}(\mu_0, \ldots, \mu_{i-1})}{\mathsf{E}[\sigma_0^2, \ldots, \sigma_{i-1}^2]}, \tag{10}$$

with the class means μ_i and variances σ_i^2 for all possible values of the variable, where i is the class label. This method only works in the profiled setting. The advantage of the SNR is that leakage from independent variables is additive. Therefore, it allows to determine the contribution of different variables to the overall leakage. Typically the number of classes is high, e.g. 256 if the target is a byte value, so a large number of traces is required to calculate a sufficient estimate of the means and variances.

Instead of SNRs, we mainly use two leakage tests with a direct statistical interpretation. During the initial search for leakage in the full side-channel traces or when the measurement position is to be determined by evaluating many possible positions, we prefer to use the t-test. For a more precise investigation to assess the amount of exploitable leakage and in order to determine POIs we use the so-called 'correlation-based leakage test' described by Durveaux and Standaert [DS16]. It has the advantage over different forms of t-tests (e.g. CRI's non-specific fixed vs. random t-test [GGJR$^+$11]) that the metric, and the POIs identified with it, directly translate to the success rate of the attack. We refer to this correlation based leakage test as CPOI.

A.2.2 Leakage assessment based on the t-test

The t-test can be easily adapted to test for all kinds of leakage, like plain- or ciphertext, key and subkeys, and any intermediate values. For example, to test for plaintext leakage using a t-test, a set of measurements with fixed plaintext and key is compared to a set with random plaintexts and fixed key using their respective mean values or other statistical moments. The t-test outputs so-called t-values which indicate statistically significant differences in the statistical moments. The significance threshold is usually set to the value 4.5 [SM15], which corresponds to a confidence level greater 0.99999. The absolute t-values can be compared and used directly as scores to determine good measurement positions. However, they cannot be used to gain insights into the success probability of attacks. T-tests are especially useful if the number of available measurements is low, since statistical moments need to be estimated only for two classes. This is most interesting in cases where the measurement position is unclear. Compared to the CPOI leakage test, a smaller, more feasible number of traces needs to be recorded at every position.

We use the t-test to search for key leakage and identify possible executions of the key schedule. To test for key leakage, a set of measurements with fixed key and random plaintexts is compared to a set with random keys and random plaintexts.

A.2.3 Univariate correlation-based leakage test (CPOI)

The correlation-based leakage test was initially proposed as a leakage test which does not require key knowledge. It uses plaintext bytes as a base to separate all measurements into different classes for the generation of univariate profiles (mean traces). These profiles are then correlated (Pearson correlation) in an univariate manner with the attack set to generate a correlation trace. The test detects the leakage of the plaintext byte itself as well as the leakage of all intermediate byte values which have a bijective relation to this plaintext byte (e.g. values after key addition or s-box substitution). The result of a correlation-based leakage test is a correlation coefficient and directly implies the success-rate of an actual univariate profiled DPA attack.

The correlation-based leakage test as described by Durveaux and Standeart includes statistical cross-validation through repeatedly segmenting all available measurements into a larger profiling and smaller disjunct attack set (with all attack sets being disjunct) to minimize statistical estimation errors of the Pearson correlation coefficient. The profiling set is then used to estimate the leakage for a DPA attack based on the Pearson correlation coefficient. The results from multiple repetitions using the different attack sets are averaged to complete the cross-validation. The leakage test effectively equals a profiled univariate CPA attack with cross-validation.

This leakage test can also be targeted at other variables like transitions between intermediate values in storage registers (i.e. Hamming distances) to focus on the leakage of only these intermediate values. In this case, there is no possible bijective relation to other values, and consequently no false positives.

A.2.4 Use cases and limitations

Both, t-tests and correlation-based leakage tests can generally be performed in a setting with either unknown, or known keys. A setting with unknown but fixed keys allows to find plaintext and ciphertext leakage and dependent leakages. A setting with known and changeable keys also allows to identify key schedule leakage.

Considering AES as an example, we observe that a known plaintext byte and an intermediate value after key byte addition and the substitution-box layer (sub-Bytes) have a bijective relation. The bijective property means that they represent a permutation (or 'relabeling') of the input which has no effect on the test. Therefore, both tests may identify leakage due to the plaintexts as well as key-dependent intermediate values. Thus, intermediate values, for example after key addition and after byte substitution, cannot be distinguished from direct leakage of the plaintext. This non-distinguishability continues until after the first mix-columns, where three other (not modelled/profiled) bytes mix into the intermediate values and the bijection no longer exists. Hence, one usually has to sort out POIs where only plaintext leakage is present. This is done by using a more specific leakage model (as for example the Hamming weight) that deliberately breaks the bijection. Another option is to take measurements where both key and plaintext vary, if the device allows this. However, the specific leakage model may not be accurate. In such a case it is preferable to perform a test based on original values. When targeting leakage from the key schedule, no false positives are expected.

A.2.5 Multivariate leakage assessment

Leakage is generally multivariate and attacks based on multivariate leakage (e.g. profiled multivariate DPA) are superior to univariate approaches like the implicitly univariate correlation-based DPA, which has been popular for a long time.

Bruneau et al. [BGH+15] as well as Unterluggauer et al. [UKM+18] describe how to compute multivariate SNRs. Both provide a formula to estimate multivariate SNR from signal and noise covariance matrices. Bruneau et al. derive an approximation in the limit of many POIs through the calculation of an LDA, while Unterluggauer et al. present an approximation in the limit of many traces using a mutual information estimate. In addition, Bruneau et al. show the effect of autoregressive noise on the SNR. Autoregressive noise means that adjacent time-samples are affected by correlated noise, which is realistic for high-frequency sampling of physical dimensions where a certain low-pass behavior always has to be assumed.

Importantly, multivariate leakage assessment is not necessary to select POIs from a trace for the obvious reason that every time-sample is assessed independently. However, strictly speaking, one should aim at determining measurement positions, which maximize multivariate leakage. But in many cases, it may be practically sufficient to select measurement positions based on univariate leakage assessments. For this, it is helpful to choose a suitable measure to combine results from univariate leakage tests into a single score. This can for instance be done by using the maximum or the averaged/summed univariate correlations. This simplification may be useful, even though it is obviously inaccurate since the assumption behind a summation of SNRs is that the noise in all univariate samples is independent/uncorrelated (see Bruneau et al. [BGH+15]). The assumption behind taking the maximum univariate SNR would be that the signal and noise are strongly correlated over all POIs (i.e. approximately equal), and there is little gain in the multivariate evaluation over the univariate one. The reality lies somewhere in-between, since noise is usually slightly correlated.

A.3 Dimensionality reduction

Bruneau et al. [BGH+15] confirm the common belief of experts in the field of pattern classification that in a profiled setting, linear discriminant analysis (LDA) is the optimal method for dimensionality reduction. Hence, principal component analysis (PCA) can be disregarded in this profiled setting. They also confirm that the success rate of optimal attacks is ultimately equal before and after LDA. This means that LDA can be used if the number of dimensions impairs computation speed of attacks. However, LDA as well as estimating profiles for template matching require the computation of class covariance matrices. Hence, if the number of POIs (n) to be processed becomes 'too large' (matrices $n \times n$), LDA does not help because the same covariance matrix has to be computed. A preliminary selection of POIs based on a univariate leakage test is usually performed for this reason.

The use of trace compression by other means such as e.g. averaging n consecutive time-samples depends on the specifics of the device and recording setup. Oversampling could reduce noise components with higher frequencies than the actual signal but is not usually employed according to the current state-of-the-art.

A.4 Template attacks

The goal of template attacks is the classification of key-dependent intermediate variables or subkeys during a cryptographic computation. As a preliminary step, in many cases, POIs are selected from the measurement or a dimensionality reduction is performed as described above. This section describes the approach based on state-of-the-art algorithms to classify individual values during profiling and the computation of probabilities for subkey candidates during the attack.

A.4.1 Recovering individual subkeys and computing discriminant scores

It is usually assumed that an exploitable signal from a variable v in a device has a multivariate Gaussian distribution with signal means $\boldsymbol{\mu}$ and additive Gaussian noise with covariance matrix $\boldsymbol{\Sigma}$. The observed noise is combined from electrical and switching noise. Switching noise commonly refers to the side-channel signal of other processed values in a digital device. The various sources of noise and their characteristics are discussed at length in the book of Mangard et al. [MOP08]. During a profiling phase the distribution parameters of the multivariate Gaussian distributions are estimated for every possible value of v using many side-channel observations $\mathbf{t}$. This allows to compute the probability density function for observing a certain measurement $\boldsymbol{t}$ given the respective value v based on the estimated parameters $\boldsymbol{\mu}_v, \boldsymbol{\Sigma}_v$ as

$$p(\boldsymbol{t}|v) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\boldsymbol{\Sigma}_v)}} \cdot \exp\left(-\frac{1}{2} \cdot (\boldsymbol{t} - \boldsymbol{\mu}_v)' \cdot \boldsymbol{\Sigma}_v^{-1} \cdot (\boldsymbol{t} - \boldsymbol{\mu}_v)\right). \tag{11}$$

The variable v cannot be observed directly and is used only to identify different classes. Therefore, it may be replaced by any unique choice of class labels.

During the evaluation of a trace in the attack phase the observation is compared to the parameterized distributions of different values v. Given the probability density function above and applying Bayes' theorem, the likelihood $L(v|\boldsymbol{t}) = p(v|\boldsymbol{t})$ can be computed. By finding the argument v that maximizes the likelihood, we obtain the most likely candidate

$$v^* = \underset{v}{\operatorname{argmax}}\, L(v|\boldsymbol{t}).$$

Choudary and Kuhn [CK13] describe several simplifications for the computation of this maximum likelihood by dropping the normalization. The result of applying Bayes' theorem is first simplified by removing the denominator which is equal for all values and by removing the a priori probabilities of values v since they are assumed to be equally likely, such that the likelihood is proportional to the profiled distribution function

$$L(v|\boldsymbol{t}) \propto p(\boldsymbol{t}|v).$$

By calculating the logarithm of the likelihood, numerical stability is improved. Since these operations are monotone functions, the order of the log-likelihood scores $d_{\log}(v, \boldsymbol{t}) = \log L(v|\boldsymbol{t})$ and in particular its maximum is unchanged.

Usually, it suffices to assume that the noise is not correlated with the value of the variable v and we can replace $\boldsymbol{\Sigma}_v$ by the pooled covariance matrix $\boldsymbol{\Sigma}_{\text{pooled}}$, which is the average of covariance matrices over all classes. According to Choudary and Kuhn [CK13] it is always recommended to try pooled covariances first. Pooling improves the estimate of the covariance matrix, which helps numerical stability during the required inversion. This leads to the simplified expression for the discriminant score $d_{\log}$ of an observation $\boldsymbol{t}$ belonging to a certain value v

$$d_{\log}(v, \boldsymbol{t}) = -\frac{1}{2} \cdot (\boldsymbol{t} - \boldsymbol{\mu}_v)' \cdot \boldsymbol{\Sigma}_{\text{pooled}}^{-1} \cdot (\boldsymbol{t} - \boldsymbol{\mu}_v) \tag{12}$$

and

$$v^* = \underset{v}{\operatorname{argmax}}\, d_{\log}(v, \boldsymbol{t}). \tag{13}$$

The factor $-\frac{1}{2}$ is kept to ensure that later, probabilities can be computed from the results (see Section A.4.3). Equation (12) is equal to Equation (23) in [CK13].

Choudary and Kuhn [CK13] also describe further simplifications by expanding Equation (12) and omitting the quadratic term in $\boldsymbol{t}$, which is constant for all values v due to the pooled covariance. They achieve significantly faster computation and more robust results in their empirical evaluation.

A.4.2 Combining multiple traces

Typically an attack includes the evaluation of multiple traces $\boldsymbol{T} = \{\boldsymbol{t}_i\}$. Choudary and Kuhn [CK13] describe how likelihoods can be estimated from multiple traces. We use Equation (28) in [CK13], which calculates the joint likelihood as the product over all traces, which leads to summation in the logarithmic discriminant score,

$$d_{\log}^{\text{joint}}(v, \boldsymbol{T}) = -\frac{1}{2} \sum_{\boldsymbol{t}_i \in \boldsymbol{T}} (\boldsymbol{t}_i - \boldsymbol{\mu}_v)' \cdot \boldsymbol{\Sigma}_{\text{pooled}}^{-1} \cdot (\boldsymbol{t}_i - \boldsymbol{\mu}_v). \tag{14}$$

Choudary and Kuhn [CK13] also describe the alternative option of computing the joint likelihood by first averaging the traces and then computing the discriminant score from the averaged traces. Although this is computationally much faster, in general it leads to too low estimates of the covariances. However, with pooled covariance matrices the averaging of traces leads to exact scores, which they confirm through empirical results. The linearized score for multiple traces then becomes

$$d_{\text{linear}}^{\text{joint}}(v, \boldsymbol{T}) = |T| \left(\boldsymbol{\mu}_v' \cdot \boldsymbol{\Sigma}_{\text{pooled}}^{-1} \cdot \bar{\boldsymbol{t}} - \frac{1}{2} \boldsymbol{\mu}_v' \cdot \boldsymbol{\Sigma}_{\text{pooled}}^{-1} \cdot \boldsymbol{\mu}_v \right), \tag{15}$$

where the average over all traces is

$$\bar{\boldsymbol{t}} = \frac{1}{|T|} \sum_{\boldsymbol{t}_i \in \boldsymbol{T}} \boldsymbol{t}_i$$

and $|T|$ denotes the number of traces.

A.4.3 Retrieving probabilities from discriminant scores

Rank estimation and enumeration algorithms require probability estimates for each hypothesis as input. Since the computed discriminant is not a probability, we need to derive a probability from it to be able to use the key rank estimation algorithm. This is done by inverting the logarithm through exponentiation and normalizing the scores to sum to one, which recovers the probability densities $p(v|\boldsymbol{t})$.

A.4.4 Choosing subkeys

Usually, a complete secret is recovered through repeating a template attack on different parts of a key-dependent intermediate values. This segmentation into parts of certain bit-lengths which are modeled and classified at once (e.g. 8 bit) is frequent and often lacking reasoning. For example in the case of AES, due to the algorithmic structure, 8 bits are the obvious choice because all operations are byte-wise. The upper limit for the bit-length of the attacked subkeys is given through computational and memory constraints as well as through the number of traces available for estimating the statistical moments of the templates.

In some cases, one set of templates can be used to attack all parts of the secret since their leakage behavior is identical. In other cases, different sets of templates are built for different parts of the secret.

Divide and conquer assuming independent variables and leakage. Most commonly, a simple divide and conquer approach is employed to recover all parts of the key. This approach is the least complex to implement. Parts of the secret are classified through independent template attacks on key-dependent intermediate values, and subsequently combined through key enumeration or rank estimation algorithms. One main assumption for attacking parts of the key independently is that the noise and signal of different parts are not correlated. This may be the case if a software implementation handles individual values sequentially and with significant time gaps between them. Also, current key rank estimation and enumeration algorithms assume *independence* of the variables, which is true under the above assumptions.

Hence, if there is no other relation between key parts (i.e. they are chosen independently at random) and their leakage is not dependent, the current algorithms may be used without further consideration (see Section A.5 about key rank estimation). See Section 3.1 for more discussion in this direction. As always, if assumptions are not met precisely, it may still be reasonable to proceed if satisfying results are achieved.

If divide and conquer does not apply - dependent variables. In case that the leaking variables are dependent due to their algorithmic relations or specifics of the implementation, it is more difficult to derive marginal probabilities. In Section 3.1 we describe solutions for the different cases of variable and leakage dependencies.

In case dependencies cannot be resolved otherwise, classified variables and their dependencies can be modeled as graphs. Then, probabilities of observations can be propagated through the graph according to their mutual dependencies using dedicated belief propagation algorithms.

Such approaches also help to exploit the leakage of many intermediate values with known functional dependencies. Veyrat-Charvillon et al. [VCGS14] describe this as Soft-Analytical SCA (SASCA) and presented it as an improvement over previous so-called *algebraic side-channel attacks* which use solvers (e.g. SAT solvers). Many intermediate values including their algorithmic dependencies are modeled through a factor graph. Algorithms from information theory, like belief propagation algorithms, are used to combine the probability information from different values to derive better probabilities on key parts. Grosso and Standaert [GS14] compare SASCA to algebraic attacks and to the straightforward divide and conquer approach. They conclude that SASCA requires less traces if the implementation leaks additional intermediate values with known functional dependencies.

However, even after propagating beliefs, the correct key may not be the most likely. Hence, an attacker needs key enumeration to find the correct combination of parts based on the updated probabilities. Belief Propagation can therefore be used to recover marginal probabilities of variables that are independent but not directly observable through leakage. Usually for any encryption scheme the secret is chosen uniformly at random and it should be possible to achieve a suitable representation. Since currently published (optimal) key enumeration algorithms assume independent variables as input, they can be applied directly. However, information loss must be expected for cases where the secret is not represented by independent variables and consequently the marginals remain dependent.

Simultaneous leakage. Another reason to look into more complex algorithms for the classification of individual parts and the combination to derive a final secret could be if parts of the secret leak simultaneously. Then, the leakage of different parts cannot be observed independently because the template attack is affected by switching noise from the unprofiled parts of the key, i.e. the unprofiled key values generate signals which act as noise. If the number of bits with overlapping signals in the side-channel observation is small enough, they can be classified at once. In this manner, the leakage signal of all bits is exploited together and the effect of switching noise prevented.

If it is not possible to profile the entire secret at once, a selection of bits should be made by attempting to maximize SNR or correlations of subkeys. How parts of a secret can be classified properly and combined to an entire secret, including a possibility for enumeration, highly depends on the kind of leakage of the individual parts. See Section 3.1 for more discussion on how the kind of leakage affects the approach to achieve key enumeration.

A.5 Key rank estimation

The previously described template attacks provide classification probabilities for candidates of subkeys to match the observed traces. All popular and recent contributions assume *independence* of subkey probabilities. From lists of probabilities for subkey candidates, they allow to enumerate the key candidates in an optimal order based on the combined classification probability for the entire key. The number of trials to find the correct key is called the key rank. The logarithm of the key rank is the security level in bit and describes the remaining entropy of the key after the attack. For security evaluations it is usually sufficient to estimate the security level directly without enumerating the keys. There are several proposals for rank estimation algorithms based on estimating the joint probabilities of the subkeys using histograms, starting with the work of Veyrat-Charvillon et al. [VCGRS12]. We use a convolution based variation of the algorithm by Glowacz et al. [GGP$^+$15]. An alternative approach is based on path counting and was proposed by Martin et al. [MOOS15]. In their follow-up papers [MMO18], [MM18], they argue that both approaches are mathematically similar and optimal. Clearly, due to the algorithmic differences, they still differ in computational and numerical performance. Martin and Martinoli [MM18] also show how path counting and convolution algorithms can be parallelized.

Li et al. [LWWW17] describe a key enumeration (not rank estimation) algorithm which handles multiple resulting score lists from different side-channel attacks for the same key bits. For example they combine recovered key byte probabilities from a regular CPA with recovered byte XOR-difference probabilities from a correlation-enhanced collision attack. Those two are obviously dependent and common enumeration algorithms are unable to exploit both results jointly. An alternative approach in such situations is to propagate probabilities (beliefs) for the same subkeys within a factor graph to derive marginal probabilities for variables, which are independent.

A.6 High-precision EM measurement setup

We use a Langer ICR HH high-precision EM probes with diameters 500 µm, 250 µm, 150 µm, and 100 µm. In addition to the built-in 30 dB amplifier of the probe, another Langer PA303 30 dB pre-amplifier is employed. We use a LeCroy WavePro 725Zi oscilloscope with 2.5 GHz bandwidth and a sampling rate of 5 GS/s, 2.5 GS/s, or 1 GS/s. We take measurements in a grid, with step sizes of 50 µm to 200 µm.