IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 4, pp. 460–492. DOI:10.46586/tches.v2023.i4.460-492

Pincering SKINNY by Exploiting Slow Diffusion

Enhancing Differential Power Analysis with Cluster Graph Inference

Nicolas Costes and Martijn Stam

Simula UiB, Bergen, Norway
nmr.costes.pro@gmail.com,martijn@simula.no

Abstract. Lightweight cryptography is an emerging field where designers are testing the limits of symmetric cryptography. We investigate the resistance against side-channel attacks of a new class of lighter blockciphers, which use a classic substitution–permutation network with slow diffusion and many rounds.

Among these ciphers, we focus on SKINNY, a primitive used up to the final round of NIST's recent lightweight standardisation effort. We show that the lack of diffusion in the key scheduler allows an attacker to combine leakage from the first and the last rounds, effectively pincering its target. Furthermore, the slow diffusion used by its partial key-absorption and linear layers enable, on both sides, to target S-Boxes from several rounds deep.

As some of these S-boxes leak on the same part of the key, full key recovery exploiting all leakage requires a clever combining strategy. We introduce the use of cluster graph inference (an established tool from probabilistic graphical model theory) to enhance both unprofiled or profiled differential power analysis, enabling us to handle the increase of S-Boxes with their intertwined leakage.

We evaluate the strength of our attack both in the Hamming weight model and against two implementations running on an STM32F303 ARM Cortex-M4 hosted on a ChipWhisperer target board, showing that our attack reduces the number of traces required to attack SKINNY by a factor of around 2.75.

Keywords: Lightweight Cryptography · SKINNY · Belief Propagation · Differential Power Analysis · Cluster Graphs

1 Introduction

The last decades have seen the emergence of the Internet-of-Things (IoT), an ever-increasing number of appliances, sensors, and control systems communicating wirelessly with their networks. This influx of highly constrained, often low powered devices has prompted the rise of a new research topic: lightweight cryptography. As existing standards are too "heavy" for IoT devices, lightweight cryptography aims to design new algorithms that provide adequate security while being cheaper to run. This objective motivated their designers to explore exciting alternative design strategies to, for instance, the current standard AES [AES01].

In 2018, the National Institute of Standards and Technology (NIST) started a standardisation process for lightweight Authenticated Encryption with Additional Data (AEAD) schemes [NIS18]. The first rounds of this process focused on traditional cryptanalysis. However, in the final round, NIST gave particular attention to Side-Channel Attacks (SCA) and their countermeasures [NIS22]. Several candidates relied on (for example) lightweight blockciphers [BCI+21, GIK+21, BJK+20], raising the question of how these ciphers fare against SCA.

Received: 2023-04-15 Accepted: 2023-06-15 Published: 2023-08-31

Heuser et al. [HPGM20] showed that various lightweight S-Boxes have an SCA resilience comparable to that of the AES S-Box. Yet, they also observed that the lack of a proper key schedule in the LED cipher [GPPR12] does lead to a trivial security degradation: an attacker can combine S-Box leakage from the first and the last round, potentially granting it twice as many S-Boxes to target. Many lightweight ciphers keep the Substitution–Permutation Network (SPN) paradigm of AES, but with lighter, less conservative components, including the aforementioned S-boxes, as well as the diffusion layer, the key schedule and even key injection. Consequently, more rounds are needed to achieve security against linear and differential cryptanalysis [BKL+07, GPPR12, BPP+17]. A natural follow-up question is how these lightweight aspects, especially slow diffusion and simplified key-scheduling, can be exploited by an SCA attacker, and which existing and new techniques are beneficial to leverage those lightweight aspects.

SCA and Blockciphers Side-channel attacks can be categorized depending on whether they are profiled or not, where the profiling refers to an adversary's ability to learn about a device with full control over its inputs, including those inputs that are usually secret or internal, such as keys and randomness used for masking. Profiling attacks are more powerful (and the default for certification), yet non-profiled attacks remain relevant. Azouaoui et al. [ABB+20] make a further classification between simple approaches and advanced ones.

Simple approaches include well-understood divide-and-conquer distinguishers (and their extend-and-prune generalization), where an adversary partitions the full key into smaller subkeys (typically bytes for AES-128) and recovers each subkey independently of the others. These distinguishers can be either profiled (e.g. template attacks based on the maximum likelihood principle) or non-profiled (e.g. correlation power analysis) and benefit from being very efficient, even for large numbers of traces, with low memory consumption. However, their downside is that they only exploit leakage from intermediates that rely on those smaller subkeys. Consequently, these simple approaches can typically only target the input and output of the S-Boxes either in the first or last round, making them less suited to exploit slow diffusion (which only manifests itself for later rounds).

Advanced approaches can look deeper into the cipher and include for instance algebraic attacks [RS10] and attacks utilizing belief propagation [VGS14]. For these attacks, an adversary tries to predict, for every trace, the values of a large number of interconnected intermediates and subsequently consolidate the belief of all those intermediates across the traces to derive the full key. The downside is that accurate prediction of all the intermediates ostensibly requires profiling. Moreover, the consolidation across traces and intermediates tends to be more memory and computationally intensive than the aforementioned simple approaches.

Lightweight ciphers with slow diffusion will lead to an abundance of leakage and currently the simple approaches cannot exploit said leakage, whereas the advanced approaches require profiling and are resource intensive. Thus, a natural question is whether an attacker can make optimal use of additional leakage while scaling even to a large number of traces, and possibly without the need to profile.

Our Contribution We propose a novel approach that fills the gap between the simple and advanced approaches. Our approach retains the computational complexity of traditional DPA, scaling linearly in the number of traces with constant memory, and works for both profiled and non-profiled attacks. Following the direction of multi-target DPA introduced by Mather et al. [MOW14], we start by building score vectors for each S-Boxes before combining them. We then reframe the problem of combining scores using the concept of factors and especially cluster graphs, known from probabilistic graphical model theory [KF09]. This reframing allows us to leverage that field's established tools for Cluster

Graph Inference (CGI), specifically clique trees and a max-product algorithm, to obtain the maximum a posteriori distinguisher on the key from our scores.

To confirm the practicality of our CGI enhancement, we focus on SKINNY, the underlying primitive of a NIST finalist [BJK+16, IKMP20a]. SKINNY is a cipher that pushes the slow diffusion approach to the envelope, making it a good choice to study how well our CGI enhancement takes advantage of lightweight features. Its designers chose a "very sparse diffusion layer and a very light key-scheduler" with "locally non-optimal internal components". It has been extensively evaluated against traditional and differential cryptanalysis, but so far has received little attention from the SCA community.

We start by analysing SKINNY from the point of view of an SCA attacker (Section 3). Its tweakey schedule is so light that it completely lacks diffusion between rounds: every byte of every round key depends only on a single byte of the master key. This characteristic leads to a situation similar to LED, where an attacker with access to both plaintext and ciphertext can combine leakage from the first and last few rounds, effectively pincering the cipher to recover the full key.

For the permutation layer, our attention remains on diffusion. SKINNY's MixColumns operation uses a sparse binary matrix, contrasting with the maximum distance separable approach used by AES. As only the top two rows of the internal state receive some round key material, the diffusion of the key into the state is shallow. The practical consequence for SCA is that many intermediate variables over multiple rounds depend on fewer than 32 bits of the master key. If the attacker can deal with subkeys of this size, then 70 S-Boxes from 6 different rounds can be targeted. In this work, we limit ourselves to S-Boxes that depend on at most 16 bits of the key, giving us 44 S-Boxes to use.

For experimental validation, we use CGI to combine scores in six distinct scenarios involving unprotected SKINNY, varying both the trace acquisition and the distinguisher producing the raw scores. Specifically, we consider a maximum-likelihood-estimate (MLE) distinguisher against the noisy Hamming weight (HW) model [MOP07, 3.3.2], for two different noise levels. Furthermore we run both a profiled MLE "template" distinguisher and an unprofiled correlation "CPA" distinguisher [BCO04] against two different implementations of SKINNY running on an ARM-Cortex M4 (Section 5). To illustrate the efficacy of our attack, in all cases we compare it to a corresponding attack that uses only one S-Box per key byte (as textbook DPA against AES would). Figure 1 shows, for the four different MLE scenarios, the reduction in the number of traces required to reach a target success rate compared to the baseline of exploiting leakage of only one S-box per key byte. Some of the graphs are only partial as the number of traces required for the baseline to reach a higher success rate exceeded our available resources. For the same reason, we excluded CPA from Figure 1, as the baseline never reached a meaningful success rate. Before running the attack, we reckoned that the ratio of the number S-Boxes targeted, namely $44/16 = 2.75$, could be indicative of our new attack's potency. Our experiments confirm this value (within experimental margins) both for the HW model and the implementation that uses lookup tables (LUT) for the S-Box computation. The other implementation is a clear outlier, as discussed in Section 5.

Our results demonstrate that slow diffusion can be exploited by an SCA attacker. Thus, if SCA is part of the threat model, ciphers using a light design strategy (similar to SKINNY) might require more protection than less light ciphers and a rigorous, fair comparison of protected design is necessary, as the lightness of a cipher may not automatically carry over to a protected implementation.

Related Work Several works have already examined the strength of lightweight ciphers against SCA and highlighted some of their exciting features. Banciu et al. [BOW15] investigate three ciphers (PRESENT, LED and KLEIN) and compare them to AES using pragmatic simple power analysis [Man03] (essentially key enumeration exploiting leakage

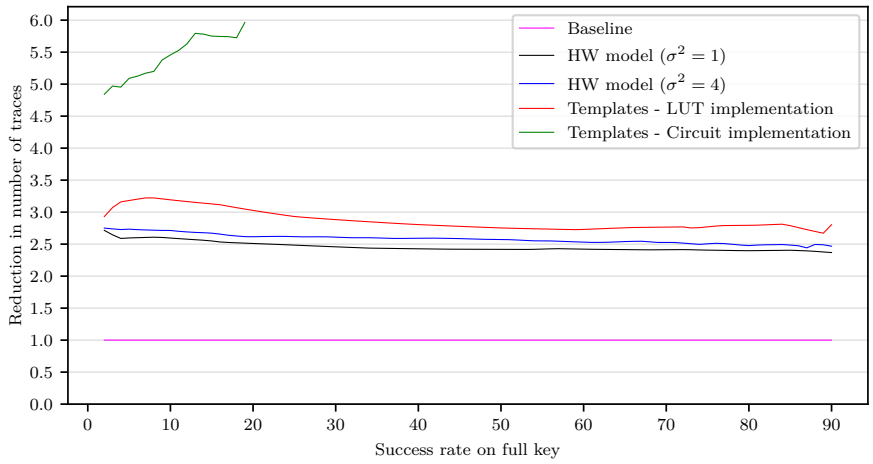


Figure 1: Reduction in the number of traces to reach a target success rate for CGI-enhanced DPA exploiting 44 S-boxes in different scenarios (for each scenario, the baseline without inference exploits only 16 S-boxes)

from a single trace). They notice that the lighter diffusion in the round function and especially the key schedule leads to more powerful attacks, as it leads to more intermediate variables whose leakage can be effectively exploited in a key enumeration effort.

Heuser et al. [HPGM20] focus on the substitution layer and compare the S-boxes of 13 different ciphers, including both nibble- and byte-oriented ones. Using both confusion coefficient and success rate as their metrics, they conclude that in the HW model, nibble-oriented S-Boxes, and more compact S-Boxes in general, are not noticeably weaker than byte-oriented ones. In contrast to our work, their results point toward lightweight ciphers being about as resilient as AES. This contrast highlights the importance of looking at the entire design when evaluating the overall resilience of a cipher against SCA.

Evaluating lightweight designs requires looking at a wide range of indicators and implementation choices that all affect the security and performance of the final cryptosystems. De Meyer [De 20] looked at the round 2 NIST candidates using metrics relevant to masking in hardware and software such as multiplicative complexity and depth. For each of those candidates, Belaïd et al. [BDM+20] generated masked implementations with guaranteed probing security, allowing a fair performance comparison taking into account countermeasures. Finally, Bellizia et al. [BBC+20] analysed the same candidates at a higher level by classifying the various components (key generation function, message computation, tag verification) based on the number of traces an adversary would realistically get access to, informing the level of protection needed.

When it comes to combining leakage from multiple sources, our work follows in the footsteps of multi-target DPA, as studied by Mather et al. [MOW14]. They combine the scores of various (non-profiled) distinguishers against AES, including ones that involve 16-bit or 32-bit key hypotheses (emanating from the MixColumns computation). However, their combinations are cleanly segregated by AES's internal columns, in the sense that the full 128-bit key can be partitioned into four 32-bit subkeys and all exploited leakage from S-boxes and MixColumns only ever affects a single of those large subkeys. In contrast, SKINNY leakage is far more interconnected, necessitating our more general approach involving cluster graphs.

Probabilistic graphical models were already introduced in the context of side-channel analysis for the recovery of passwords based on keystroke timings [SWT01] using Hidden

Markov Models (HMMs). HMMs were subsequently instrumental to mount simple power analysis attacks against elliptic curve scalar point multiplication based on randomized addition–subtraction chains, efficiently recovering the private exponent given a single trace of additions and doublings. The techniques evolved from simple linear algebra [Osw03], via a max-product algorithm [KW03], to loopy belief propagation inspired by factor graphs [GNS05].

Later, the potential of belief propagation in factor graphs was realized for profiled differential power analysis against blockciphers [VGS14] as well as profiled simple power analysis against the number-theoretic transform [PPM17]. Despite the superficial similarities, our approach using belief propagation in cluster graphs is conceptually quite different; we provide a more detailed comparison in Section 4.4.

Finally, beyond standard DPA, other methods are known that combine leakage emanating from multiple rounds of SPN ciphers. For instance, leakage from AES's second round can be exploited using collisions [BGL09] or higher-order attacks [LPdH10]. In principle, deep learning's name and heuristic nature could give the impression it might exploit the low diffusion of SKINNY, going deeper into the cipher. Yet, the typical DL-SCA workflow [BPS+20] provides no evidence it would, as we elaborate upon in Section 4.4.

2 Preliminaries

2.1 Substitution–Permutation Networks

SPN Ciphers A Substitution–Permutation Network (SPN) is one of the main paradigms to design a blockcipher [KR11]. An SPN cipher encrypts a plaintext by iterating a round function composed of a round-key addition, a substitution layer and a permutation layer.

First, a round-key derived from the master key is mixed with the input, typically using exclusive-OR. The substitution layer then splits the input into shorter blocks and applies to each block a fixed permutation, called S-Box. The resulting (short) blocks are finally mixed in the permutation layer using a carefully chosen linear function. The substitution and permutation layers achieve confusion and diffusion, respectively.

The existing standard AES [AES01] is an SPN cipher, where both the substitution and permutation layers are chosen to produce high levels of confusion and diffusion fast. In contrast, for lightweight ciphers the substitution and permutation layers are often much simpler and lighter, resulting in much slower diffusion.

Fast and Slow Diffusion Although the properties of diffusion and confusion date back to Shannon [Sha49] and Feistel [Fei70], the concept of *fast* or *slow* diffusion was not part of their original work. Yet, with the development of differential cryptanalysis it became an integral part of designing cryptographic primitives. Properties like differential branching, the minimal amount of active S-Boxes, or how many rounds are needed for full diffusion in the state, are now closely looked at by designers and standards organizations [NIS18, 4.1].

Our focus will be on SKINNY [BJK+16], part of a group of ciphers [BKL+07, BPP+17] that use very slow diffusion components, such as a sparse permutation layer and a simple key schedule. These lightweight ciphers compensate their slow diffusion with more rounds, allowing for instance SKINNY's designers to argue that their cipher still resists differential attacks. Our interest lies in how light diffusion impacts SCA and DPA in particular.

2.2 Differential Power Analysis on SPN Ciphers: AES-128

Template Attacks as Divide-and-Conquer Maximum Likelihood Estimates In a power analysis attack, an adversary tries to recover a secret key by exploiting information obtained through power measurements, usually called a trace, of a device running a cipher on known

inputs. As the power consumption of a running processor depends on the operations and the values it computes, a trace of an encryption or decryption will contain points where the key leaks, i.e. points where the power depends on the value of the key.

A well-known strategy to exploit said leakage is differential power analysis (DPA), introduced by Kocher et al. [KJJ99] and subsequently refined, including template attacks [CRR03]. Below we give a brief run through a typical divide-and-conquer template attack against AES-128 from a more modern perspective, simultaneously introducing the notation that will facilitate our new attacks against SKINNY.

An AES-128 key consists of 16 bytes, which we treat as individual random variables $K_1, \ldots, K_{16}$. The full key $\mathbf{K}$ is the set of all 16 random variables. Suppose an adversary is concentrating on the 16 first-round S-boxes, observing M traces with random known plaintexts. We use random variable X to denote the entire collection of plaintexts, L_j for the leakage (across all the traces) on S-box $j \in [16]$ and $\mathbf{L}$ for the combined leakage on all 16 S-boxes, so $\mathbf{L} = (L_1, \ldots, L_{16})$. Instantiations are denoted with the corresponding lower case, e.g. $x, \ell_j, \boldsymbol{\ell}$.

An adversary trying to optimize its success rate, would output the most likely key, also known as the Maximum-A-Posteriori (MAP) estimate

$$\arg\max_{\mathbf{k}} \Pr[\mathbf{K} = \mathbf{k} \mid \mathbf{L} = \boldsymbol{\ell}, X = x],$$

where x and $\boldsymbol{\ell}$ are the actually observed plaintexts, respectively traces, by the adversary.

Bayes's theorem allows transformation of the MAP into the Maximum-Likelihood Estimate (MLE)

$$\arg\max_{\mathbf{k}} \mathrm{p}(\mathbf{L} = \boldsymbol{\ell} \mid \mathbf{K} = \mathbf{k}, X = x),$$

as

$$\Pr[\mathbf{K} = \mathbf{k} \mid \mathbf{L} = \boldsymbol{\ell}, X = x] = \frac{\mathrm{p}(\mathbf{L} = \boldsymbol{\ell} \mid \mathbf{K} = \mathbf{k}, X = x) \cdot \Pr[\mathbf{K} = \mathbf{k} \mid X = x]}{\mathrm{p}(\mathbf{L} = \boldsymbol{\ell} \mid X = x)},$$

where we use that AES-128 keys are chosen uniformly at random and we assume that the plaintexts X to be encrypted are independent from the key. Consequently, the probability $\Pr[\mathbf{K} = \mathbf{k} \mid X = x] = 2^{-128}$ does not depend on the key guess $\mathbf{k}$, nor does $\mathrm{p}(\mathbf{L} = \boldsymbol{\ell} \mid X = x) \neq 0$, so neither influences the arg max calculation. To reflect that leakage is usually modelled using continuous distributions, we use a probability density function (pdf) $\mathrm{p}(\cdot)$ instead of a probability $\Pr[\cdot]$.

As we mentioned, we suppose that the leakage is composed of that of the 16 individual first-round S-boxes. A common modelling approximation is to assume that an S-box's leakage only depends on its input and, conditioned on that input, is independent of any other randomness in the system. Although real-world leakage is seldom so neat, we will use this approximation as well. Specifically, the leakage on different S-boxes is considered independent of each other conditioned on the full key and plaintexts, so we may instead concentrate on

$$\arg\max_{\mathbf{k}} \prod_{j \in [16]} \mathrm{p}(L_j = \ell_j \mid \mathbf{K} = \mathbf{k}, X = x).$$

A further simplification is possible by realizing that, for AES-128, the input to the jth S-box only depends on the corresponding key-byte, leading to

$$\arg\max_{\mathbf{k}=(k_1,\ldots,k_{16})} \prod_{j \in [16]} \mathrm{p}(L_j = \ell_j \mid K_j = k_j, X = x)$$

(one could also isolate the relevant bytes from the plaintexts, however as there is no conceptual benefit in doing so, we forgo introducing the notational overhead to do so).

So far, our description of DPA has been fairly standard using well-known mathematical notation common in the SCA literature. Next, we restate the final simplification

using terminology from probabilistic graphical models [KF09]. A key concept in that domain is that of a factor ϕ for a set of random variables. A factor is a map from the values over which the random variable is defined to the non-negative real numbers. For instance, a factor $\phi_1(K_1)$ maps any possible value $k_1 \in \{0,1\}^8$ to a real. Define $\phi_j(K_j) = \mathrm{p}(L_j = \ell_j \mid K_j = k_j, X = x)$ for $j \in [16]$ and let Φ be the set of ϕ_j and set

$$\tilde{P}_\Phi(K_1, \ldots, K_{16}) = \phi_1(K_1) \times \ldots \phi_{16}(K_{16})$$

then the optimization problem the distinguisher is trying to solve is

$$\arg\max_{\mathbf{k}} \tilde{P}_\Phi(\mathbf{k}) = \arg\max_{\mathbf{k}=(k_1,\ldots,k_{16})} \prod_{j\in[16]} \phi_j(k_j) \ ,$$

which equals

$$\left(\arg\max_{k_1} \phi_1(k_1), \ldots, \arg\max_{k_{16}} \phi_{16}(k_{16})\right) \ ,$$

given that each k_i only appears in a single factor ϕ_j (namely for $j = i$). This last step makes explicit the divide-and-conquer nature of a typical template or MAP attack against AES-128 as each subkey byte can be recovered independently of the others (by concentrating on the leakage of a single, relevant S-box), after which recombination is straightforward.

A common metric to evaluate how good an DPA attack works is its success rate [PGA+23]: the probability that it recovers the actual key. In that sense, the MLE is optimal, as long as all the assumptions used to derive it are met [CRR03, HRG14]. Moreover, the success rate for full key recovery equals the product of the success rates for the individual key bytes and an evaluator can easily estimate these success rates by the fraction of successful key recoveries over a sufficiently large number of experiments. We will use the success rate throughout this work to compare attacks against each other.

Another useful evaluation metric is key ranking [PGA+23] where, for a single experiment, the key rank corresponds to the position of the correct key among all key candidates (ranked from most to least likely). A global key rank histogram paints a more complete picture of an implementation's security against SCA than success rate and it can be obtained experimentally by (repeatedly) running a suitable key rank evaluation algorithm [MOOS15, YMO22] on the combined distinguishing scores for the individual subkeys (histograms for individual subkey bytes are less informative [MMOS16]). For scores emanating from SCA attacks relying on graph inference a further complication arises, as the graph inference itself could be run such that it enumerates the full keys directly (see also Step 3 of the attack outline in Section 4.4.3), which is potentially more efficient than standard enumeration based on (essentially marginalized) distinguishing scores. To the best of our knowledge, the corresponding ranking problem has not yet been studied.

Distinguishing Scores as Factors Evaluating a factor $\phi_j(K_j)$ given ℓ_j and x corresponds to the known process of calculating distinguishing scores. We can generalize this interpretation of distinguishing scores as factors as follows. A distinguisher typically targets a specific intermediate value v_j, say the output of the jth first-round S-box (as above). However, one can also target other intermediate values, for instance Mather et al. [MOW14] exploit leakage from AES's MixColumns operation, targeting as intermediate value $v_j = GFm2(state_i \oplus state_{i'})$, the doubling (in AES's finite field $\mathbb{F}_{2^8}$) of the sum of the outputs of first round S-boxes i and i'. In this case, the intermediate value v_j depends on both key bytes K_i and $K_{i'}$; whereas Mather et al. talk about distinguishing scores on 16-bit subkeys, we would instead consider those same scores as a factor $\phi_j(K_i, K_{i'})$ instead, where ϕ's subscript indicates which intermediate value is targeted and the argument which key variables are leaked upon. (In later sections, we often omit ϕ's subscript and simply identify different factors by the different key variables they leak upon.)

As factors are essentially abstracting distinguishing scores, their evaluation is standard, as it matches that of the distinguisher. We will consider the classic maximum-likelihood (MLE) and correlation (CPA) distinguishers, for which efficient algorithms are known [BB17]. In essence, for each key candidate and each trace, the known plaintext (and ciphertext) are used to predict the intermediate value and that prediction is subsequently coupled with the observed leakage to update the score for the key candidate. The time complexity of both the MLE and CPA distinguishers against byte-wise SKINNY is $\mathcal{O}(M \cdot 2^{8 \cdot W})$ per factor, where M is the number of traces and W is the (maximum) number of key bytes a factor may depend upon. The memory complexity is dominated by the storage needs of the factors, so it is $\mathcal{O}(2^{8 \cdot W})$ per factor. Crucially, as both the MLE and CPA distinguishers allow online evaluation, the number of traces only affects the data complexity, but not the memory complexity. Previous works have evaluated factors up to 32-bits subkeys [MKP12, MOW14]. Henceforth, we will call an intermediate variable (or operation) *enumerable* if its corresponding factor depends on a subkey sufficiently small to enumerate (with 32-bit subkeys or $W = 4$ as our upper limit) in order to evaluate said factor.

3 An Analysis of the SKINNY Cipher

3.1 The SKINNY Blockcipher Specification

The SKINNY Family SKINNY is a tweakable blockcipher family [BJK+16] that follows the "tweakey" framework [JNP14]. Instead of the usual key state, a tweakey state, denoted TK, contains both tweak and key. We will concentrate on the SKINNY-128 subfamily, where the tweakey state can be 128, 256 or 384 bits with 128 bits of key material, and for now, we will focus on SKINNY-128-128, meaning that TK = key. In Section 6, we will review the susceptibility to our attack of other variants of SKINNY, including SKINNY-128-384+ (as used by the latest version of Romulus [GIK+21]).

Specification SKINNY uses an SPN construction with a round function consisting of five operations: SubCells, AddConstants, AddRoundTweakey, ShiftRows and MixColumns. For SKINNY-128-128, this round function is iterated 40 times. Following the general description of an SPN cipher (Section 2), we divide the round function of SKINNY into key addition, substitution layer and permutation layer. Similar to AES, SKINNY-128's state is viewed as a bytewise 4×4 array.

The SubCells operation performs the substitution layer and uses an 8-bit S-Box applied to every byte of the state. The permutation layer uses three operations: AddConstants is an XOR of 3 constants (one fixed, two changing every round) to the first three bytes of the first column of the state; ShiftRows is similar to the AES ShiftRows, albeit shifting to the right instead of the left; finally, MixColumns is a multiplication of every state column by the binary matrix M (respectively M^{-1} for the decryption), where

$$M = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \text{ and } M^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

For its key addition, SKINNY-128-128 only takes the top two rows of its tweakey state and XORs them to the top two rows of the cipher state. (In contrast, AES-128 XORs the full round key with the full state.) Subsequently, SKINNY-128-128 uses a very light key schedule to update the tweakey state: the position of each byte in the state is shuffled using the permutation

$$P_T = [9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7] .$$

The choice of P_T has several interesting consequences. Its period is 16, so every 16 rounds $TK =$ key, which will play a role when we will look at different versions of SKINNY later on. Yet more importantly, P_T keeps the first and last 8 bytes of the key separated: K_1 to K_8 participate in odd-numbered rounds and K_9 to K_{16} in the even-numbered ones.

3.2 SCA Perspective of SKINNY

Key Observations SKINNY contains various features that provide opportunities.

(i) Firstly, the combination of only partial tweakey absorption into the state and P_T's property of keeping the two halves of the key artificially separated allow an attacker to recover the first and last half of the key independently by targeting the first and last rounds, respectively (technically, this observation also relies on an even number of rounds, which is the case for all variants of SKINNY).

(ii) Secondly, the key schedule completely lacks internal diffusion: the individual bytes in TK never interact (their positions change, but they never mix). Consequently, any given byte of any round-tweakey depends only on a single byte of the master key.

(iii) Finally, slow diffusion of MixColumns. SKINNY's designers indicate that six rounds are necessary to diffuse the key to the state fully. Unsurprisingly, such diffusion leads to multiple S-Boxes in early rounds to depend on only a few key bytes, consequently becoming enumerable. Furthermore, the two earlier features imply that the same is true for the late rounds (for an adversary with knowledge of the ciphertexts).

As we will demonstrate, the three observations above combine to allow an SCA attacker to exploit S-box leakages from both sides of the trace several rounds deep. Similar to AES-128, when attacking the first rounds, we exploit knowledge of the plaintexts, whereas when attacking the last rounds, we use the ciphertexts. As there are some subtle differences between the first and last rounds, we will consider them separately up to three rounds deep. However, our final attack will only exploit leakage from S-Boxes two rounds deep, so from 4 rounds in total (2, 3, 39 and 40).

A Look at the First Few Rounds As SKINNY starts with SubCells without whitening, the first round S-Boxes do not depend on the key and, consequently, only leak on the plaintext. Instead, we start by exploiting the S-Boxes in the second round, after the ShiftRows and MixColumns operations. Here the slow diffusion of SKINNY helps us. Consider the structure of the matrix M:

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} a+c+d \\ a \\ b+c \\ a+c \end{pmatrix}$$

and recall that only the top two rows (a and b) receive some key material (so we can imagine $c, d = 0$). Then MixColumns results in the a key bytes being spread out over three rows, gifting us multiple S-Boxes all leaking on the same a key bytes. This behaviour already constitutes a significant departure from AES, where each key byte only leaks during a single S-Box evaluation.

The light diffusion carries over to subsequent rounds, as illustrated by Figure 2. Restricting to 32-bit subkeys, all the S-Boxes in round 3 and one hapless S-Box in round 4 would be enumerable, resulting in 33 exploitable S-boxes in the first rounds. However, for

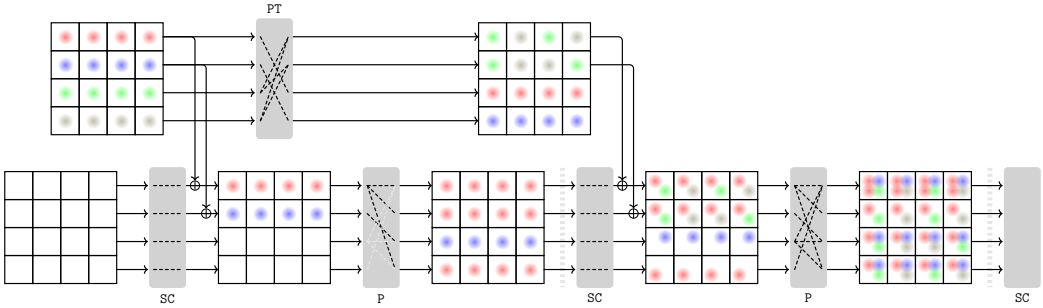


Figure 2: Key diffusion for SKINNY's first few rounds

our main attack, we limit ourselves to 16-bit subkeys, thus we can only exploit 4 S-Boxes from round 3 (in addition to the 16 from round 2, for a total of 20).

A Similar Tour of the Last Few Rounds For the exploitation of the last few rounds, it helps to think of the cipher in reverse order, almost as if it were decrypting. Indeed, for the following we will refer to cipher operations by their inverse and exploit rounds starting from the last, continuing towards the first.

The last round (40) finishes with the permutation layer, so when attacking the S-Boxes of the last round, there are no operations between AddRoundTweakey and SubCells. Since only the top two rows are affected, the two bottom rows of the state are fully determined by the ciphertext and do not yield exploitable leakage. We therefore get 8 S-Boxes that leak on K_9 to K_{16}, with each S-Box depending on a single key byte.

Similarly to the first rounds, the slow diffusion leads to exploitable S-Boxes in the 'following' two rounds. However, reversing the order of the operations results in a different pattern of how the S-boxes depend on the various key bytes. Following the SubCells from round 40, the inverse MixColumns in round 39 produces:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} b \\ b+c+d \\ b+d \\ a+d \end{pmatrix}.$$

Here again, only a and b contain key material, so MixColumns this time spreads out the b key bytes over three rows. However, there is an AddRoundTweakey before the SubCells. Thus for round 39 the S-boxes in the top two rows will each depend on two key bytes; the S-Boxes in the bottom two rows still depend on a single key byte each.

Figure 3 illustrates how the key propagates through the last rounds. For the 32-bit subkeys limit, there are 37 exploitable S-Boxes in those last three rounds. When limiting ourselves to 16-bit subkeys, we only get 24.

Initial Damage Assessment The combination of a very light key schedule and slow diffusion makes SKINNY a fascinating target for SCA, with many S-boxes depending on relatively few key bytes, as shown in Figure 4. In total, 70 S-Boxes each depend on 32 bits of the master key (or less). Even an attacker restricted to 8-bit subkeys could still use 32 S-Boxes, twice as many as for AES. For our attacks, we will exploit 44 S-boxes: 32 depending on a single key byte and 12 depending on two key bytes.

However, identifying those structural weaknesses of SKINNY does not mean that exploiting all those S-Boxes in a single attack is trivial. While a straightforward divide-and-conquer attack works for AES, and for SKINNY when restricting to 8-bit subkeys, once

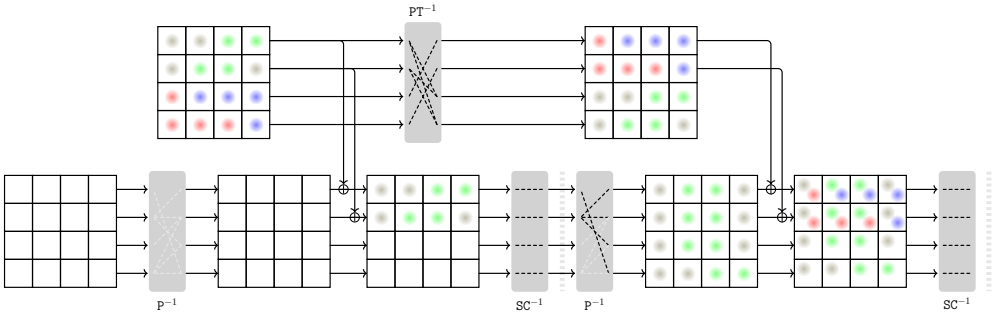


Figure 3: Backwards key diffusion for SKINNY's last few round

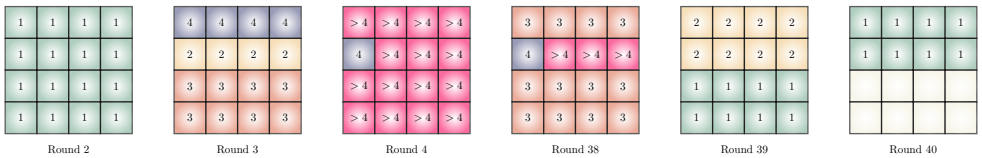


Figure 4: Key-byte dependency of each S-Box in the first and last rounds of SKINNY

16-bit subkey S-Boxes enter the fray, the same key byte can feature in multiple factors and combining those factors in an efficient and effective manner is no longer straightforward.

4 Combining Factors Using Cluster Graph Inference (CGI)

To exploit the leakage from all the S-Boxes in Section 3 we need a new strategy. For each S-Box, its leakage across traces is captured in a factor. As we restrict ourselves to S-boxes that depend on at most a 16-bit subkey, each factor depends on either one or two key bytes. Table 1 surveys which key byte combinations occur for SKINNY-128-128 and their multiplicities. We will refer to the factors from column 1 as mono-dependent factors, those from column 2 as bi-dependent factors, and finally those from columns 3 and 4 as hi-dependent factors.

In Section 2 we rephrased the key recovery task as finding $\arg\max_{\mathbf{k}} \tilde{P}_\Phi(\mathbf{k})$ where Φ is the set of all factors and $\tilde{P}_\Phi(\mathbf{K})$ is the product of all factors (with the random variables $K_1, \ldots, K_{16} \in \mathbf{K}$ correctly spread out over the factors). For AES-128, combining the factors obtained from the first round S-Boxes allowed the customary divide-and-conquer approach where each key byte is recovered independently of the others. We will next delve into how to evaluate $\arg\max_{\mathbf{k}} \tilde{P}_\Phi(\mathbf{k})$ for three new scenarios, depending on whether Φ only contains mono-dependent factors (Section 4.1), also bi-dependent factors (Section 4.2), or even hi-dependent ones (Section 4.3).

4.1 Combining Mono-Dependent Factors (8-bit Subkeys)

If we limit ourselves to S-Boxes that depend on a single key byte, i.e. the ones listed in the first column of Table 1, computing the MAP distinguisher is straightforward: we are still in a divide-and-conquer scenario. The only difference is that some key bytes leak across multiple independent S-Box computations.

Instead of having 16 factors $\phi_j(K_i)$ with $j = i$, we now have 32 factors, so some K_i feed into multiple factors, say $\{\phi_j(K_i), \phi_{j'}(K_i), \phi_{j''}(K_i)\}$. As each S-Box just depends on a single key byte, we are still guaranteed that each factor only depends on a single key byte. We can then easily combine all factors that depend on the same subkey by computing

Table 1: Summary of all available S-Box leakages up to 32-bit subkeys

8-bit subkey		16-bit subkey		24-bit subkey		32-bit subkey	
byte	#	bytes	#	bytes	#	bytes	#
K_1	3	K_1, K_{10}	1	K_1, K_6, K_{10}	1	K_1, K_2, K_6, K_{10}	1
K_2	3	K_1, K_{13}	1	K_1, K_7, K_{11}	1	K_1, K_4, K_5, K_{14}	2
K_3	3	K_2, K_9	1	K_1, K_{13}, K_{15}	1	K_2, K_3, K_7, K_{16}	1
K_4	3	K_2, K_{16}	1	K_1, K_{13}, K_{16}	1	K_3, K_4, K_8, K_9	1
K_5	1	K_3, K_9	1	K_2, K_7, K_{16}	1	$K_3, K_{10}, K_{13}, K_{14}$	1
K_6	1	K_3, K_{10}	1	K_2, K_8, K_{15}	1		
K_7	1	K_4, K_{14}	1	K_2, K_9, K_{11}	1		
K_8	1	K_4, K_{16}	1	K_2, K_9, K_{12}	1		
K_9	2	K_5, K_{13}	1	K_3, K_5, K_{13}	1		
K_{10}	2	K_6, K_9	1	K_3, K_8, K_9	1		
K_{11}	2	K_7, K_{16}	1	K_3, K_{10}, K_{14}	2		
K_{12}	2	K_8, K_{10}	1	K_4, K_5, K_{14}	1		
K_{13}	2			K_4, K_6, K_{12}	1		
K_{14}	2			K_4, K_{11}, K_{12}	1		
K_{15}	2			K_5, K_{12}, K_{13}	1		
K_{16}	2			K_6, K_9, K_{14}	1		
				K_7, K_{12}, K_{16}	1		
				K_7, K_{13}, K_{16}	1		
				K_8, K_{10}, K_{15}	1		
Total	32		12		20		6

$\psi(K_i) = \phi_j(K_i) \times \phi_{j'}(K_i) \times \phi_{j''}(K_i)$. Simply put, we can compute scores for each S-Box independently and multiply them together to obtain 16 factors for 16 key bytes.

In essence, this observation was already used by Mather et al. [MOW14], although they took the extra step of transforming their factors (or scores) into posterior probabilities. It shows that exploiting the leakage of 16 additional S-Boxes is easy for SKINNY and, as a rule of thumb, we would expect that leaking on twice as many S-Boxes means we only need half the number of traces.

4.2 Combining Bi-Dependent Factors (16-bit Subkeys)

Problem Description Combining our previous 32 S-Boxes with the 12 that depend on two different key bytes is challenging. We could exploit factors derived from such S-Boxes with marginalisation. Despite factors not being actual conditional probabilities, we can still eliminate a variable by *max marginalising* it. For example, imagine that we have a factor $\phi(K_1, K_2)$, and we wish to transform it to a factor only depending on K_1, essentially transforming a map from $\{0,1\}^{16} \to \mathbb{R}$ to $\{0,1\}^8 \to \mathbb{R}$. What we can do is, for every $k_1 \in \{0,1\}^8$, select the maximum of $\phi(k_1, k_2)$ over $k_2 \in \{0,1\}^8$. As a factor depending on two variables can be represented as a matrix, this maximization is equivalent to picking the max for each column (or row, depending on which variable to eliminate), and we denote it by $\phi(K_1) = \max_{k_2} \phi(K_1, k_2)$.

We could use this operation to exploit our additional S-Boxes. For each K_i, pick all the factors that depend on K_i, use the max-marginalisation to eliminate the other key bytes for the 16-bit subkey factors, and multiply all the resulting factors as we did before. However, this local process loses a lot of global information, and we can do much better by recovering all the key bytes simultaneously.

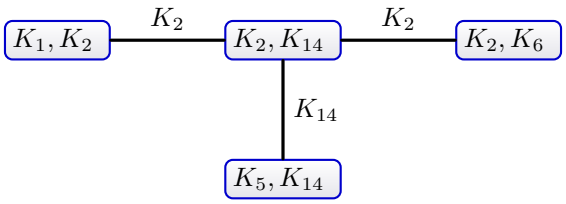


Figure 5: Example of a clique tree

Cluster Graphs and Clique Trees However, the problem we are trying to solve, namely finding $\arg\max_{\mathbf{k}} \tilde{P}_\Phi(\mathbf{k})$, can be solved much more conveniently using probabilistic graphical model theory. In particular, we will use what is called a cluster graph, adapted for factor manipulation [KF09, 10.1.1]. For a set of factors Φ over the set of variables $\mathbf{K}$, we can construct a graph where each node in the graph is associated with a subset, or cluster, $C_i \subseteq \mathbf{K}$. Factors are assigned to clusters such that all variables feeding into the factor are in C_i. For two clusters with non-empty intersections, we may draw an edge, labeled with said non-empty intersection.

For example, one can build the cluster graph given in Figure 5 for the set of factors

$$\Phi = \{\phi(K_1, K_2), \phi(K_2), \phi(K_2, K_6), \phi(K_2, K_{14}), \phi(K_{14}, K_5), \phi(K_5)\}\,.$$

Crucially, a given set of factors Φ might be represented by different cluster graphs. Firstly, there is a non-displayed choice of how factors are assigned to clusters. For the 16-bit factors there is no option but their corresponding clusters. However, an 8-bit factor can go in any cluster that contains its subkey. Concretely, for $\phi(K_5)$, only one cluster is available, but $\phi(K_2)$ could go in any of the other three (whichever is chosen does not impact the structure of the graph or our subsequent analysis and attacks).

More importantly, edges between nodes may be drawn, but there is no obligation (as long as the reduced cluster graph does not have more components than the full cluster graph). Thus, even the structure of the cluster graph is not unique, and in our example not all edges are displayed. For instance, we could have drawn the edge between (K_1, K_2) and (K_2, K_6), as they have a non-empty intersection of their clusters in K_2.

Typically, the way to construct a cluster graph is to draw all the edges initially and then remove enough edges to turn the graph into a tree. Cluster graphs are particularly efficient at solving the MAP problem if they are trees that satisfy the running intersection property (Definition 1). In that case, we call the graph a clique tree.

Definition 1 (Running intersection property (copied from [KF09, Definition 10.2]))**.** Let $\mathcal{T}$ be a cluster graph and $\mathbf{C}$ its clusters. $\mathcal{T}$ has the running intersection property if, whenever there is a variable K such that $K \in C_i$ and $K \in C_j$, then K is also in every cluster in the (unique) path in $\mathcal{T}$ between C_i and C_j.

Given a cluster graph, checking whether it is a clique tree is easy; for instance, Figure 5's graph is a tree and has the running intersection property, making it a clique tree.

A Clique Tree for SKINNY Remarkably, the 44 bi-dependent S-Boxes of SKINNY, corresponding to columns 1 and 2 of Table 1, comprise a factor set that also allows for a clique tree, as shown in Figure 6. Subkeys K_{11}, K_{12} and K_{15} are missing from the graph as they do not appear in any of the 12 S-Boxes with a 16-bit subkey. Essentially, for those subkeys the classical divide-and-conquer approach still works and their optimal guess can be recovered independently of the other subkeys (in extremis, a forest of clique trees each consisting of a single node yields the AES-128 situation).

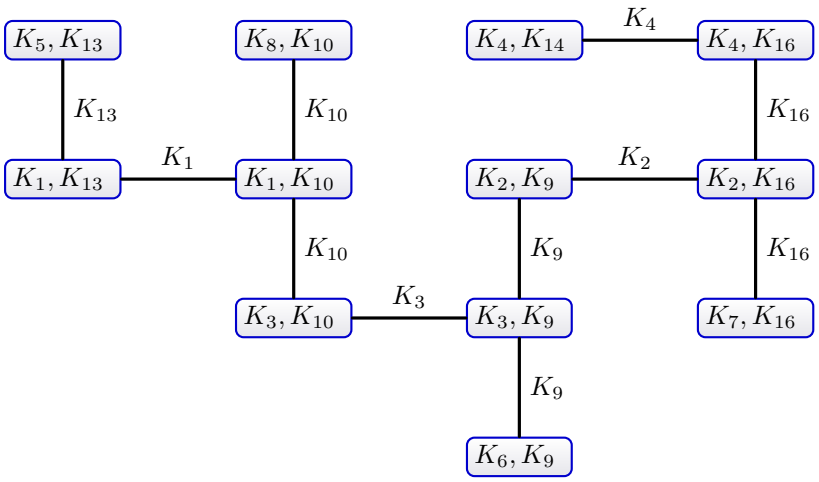


Figure 6: Clique tree formed by the 16-bit dependent leakages of SKINNY

It might seem a bit magical that the S-Boxes of a cipher arrange so nicely in a tree. Picking pairs of key bytes at random would probably lead to a loop somewhere in the graph. However, the 12 pairs are not random. From the analysis in Section 3, we recall that SKINNY's key scheduler splits the key bytes into two sets and uses them alternatingly for the round key. All our 16-bit subkeys come from round 3 and round 39 and are composed of one key byte from each set. While this construction does not guarantee that the resulting graph is a tree (for example, an extra factor $\phi(K_2, K_{10})$ would break the running intersection property), it certainly helps.

The Max-Product Algorithm As our cluster graph is a clique tree, we can use the so-called max-product algorithm (Algorithm 1) to efficiently return the exact answer to our MAP problem and effectively combine the leakage of our 44 S-Boxes. We provide a detailed explanation of how the algorithm works below, for a more in-depth explanation of why it works, see the reference work of Koller and Friedman [KF09, Algorithms 10.1, 10.2 and 10.3].

Algorithm 1 takes as input a set of factors Φ, a set of clusters $\mathbf{C}$, and a mapping α from the factors to the clusters (where the convention is that the clusters are indexed by natural numbers). In the case of our attack on SKINNY, the set of factors are the distinguishing scores resulting from the mono- and bi-dependent S-boxes (first two columns of Table 1) and each factor over two subkeys (second column of said table) goes into its own cluster, where the assignment of the factors over a single cluster to any cluster having that subkey in its scope can be arbitrary (it will not affect the result of the algorithm). The algorithm additionally takes as input a clique tree $\mathcal{T}$ (Figure 6 for SKINNY) with one arbitrary cluster C_r in the clique tree designated as root (again, the choice does not affect the eventual result).

During initialization, for each cluster C_i, the initial belief β_i for its variables is set as the non-empty product of the factors assigned to it:

$$\beta_i \leftarrow \prod_{\phi\colon \alpha(\phi)=i} \phi\,,$$

where the product is essentially a Hadamard-style product. For instance, in the case of SKINNY, imagine the root cluster C_r has the three factors $\phi_1(K_1, K_{10})$, $\phi_2(K_1)$ and $\phi_3(K_{10})$ assigned to it (so $\alpha(\phi_1) = \alpha(\phi_2) = \alpha(\phi_3) = r$). Then for all $k_1, k_{10} \in \{0,1\}^8$, we would calculate $\beta_r(k_1, k_{10}) = \phi_1(k_1, k_{10}) \cdot \phi_2(k_1) \cdot \phi_3(k_{10})$.

As the algorithm uses message passing, each edge (i, j) in the clique tree $\mathcal{T}$ is associated with a message previous $\mu_{i,j}$ that was sent over that edge. All messages $\mu_{i,j}$ are initialized to 1, the neutral element under multiplication. As edges have no direction, we stipulate that $\mu_{i,j} = \mu_{j,i}$ by convention. Yet, when new messages are sent, they do flow from one cluster to another, as indicated by the notation $\delta_{i\to j}$ (see below).

The algorithm proper proceeds in two phases of message passing: in the upwards pass, belief is coagulated towards the root and in the subsequent downwards pass, it is spread back across the tree to ensure consistent beliefs throughout. During the upwards pass, each non-root cluster C_i will transmit once, and it can do so once all-but-one of its neighbours have transmitted, when C_i's message will be to that exceptional neighbour C_j that has not transmitted yet. Thus, the upwards pass starts at the leaves of the tree and ends at the root (which does not send a message in the first phase).

The message transmitted between C_i and C_j, denoted $\delta_{i\to j}$, is the current belief held by C_i max-marginalized by the subkey in C_i not on the edge of transmission, i.e. the subkey not in the intersection of C_i and C_j:

$$\delta_{i\to j} \leftarrow \max_{C_i - C_i \cap C_j} \beta_i \,.$$

For instance, for SKINNY's clique tree, when the cluster C_i with only the factor $\phi_1(K_8, K_{10})$ messages the cluster C_j with factor $\phi_2(K_1, K_{10})$, the max-marginalization ensures that the message sent is about the belief C_i holds about K_{10}, as is relevant to C_j. Specifically, for each $k_8, k_{10} \in \{0,1\}^8$ we have that $\delta_{i\to j}(k_{10}) \leftarrow \max_{k_8} \phi_1(k_8, k_{10})$, where $\beta_i = \phi_1$ as C_i contains only that single factor.

The receiving cluster then incorporates the message in its belief with the product

$$\beta_j \leftarrow \beta_j \times \frac{\delta_{i\to j}}{\mu_{i,j}} \,.$$

For the example above, C_j would compute, for each $k_1, k_{10} \in \{0,1\}^8$, $\beta_j(k_1, k_{10}) \leftarrow \beta_j(k_1, k_{10}) \cdot \delta_{i\to j}(k_{10})$, where we omitted the division by $\mu_{i,j} = 1$.

Finally, $\mu_{i,j} \leftarrow \delta_{i\to j}$ reflects that the last message transmitted on the edge (i, j) has changed. As a small aside, as each $\mu_{i,j}$ is only used once in the upwards pass, one could dispense with its initialization, omit the division by $\mu_{i,j} = 1$ when calculating $\delta_{i\to j}$ and instead rely on the $\mu_{i,j} \leftarrow \delta_{i\to j}$ update to have well-defined $\mu_{i,j}$ for the subsequent downwards pass; we opted for the current write-up to highlight the symmetry between both passes.

Once the root is the only cluster left transmit a message, the downwards pass begins. A cluster is called *informed* if it has received the information from all the other clusters, and at this stage, the only informed cluster is the root cluster. A second phase of message passing begins, from informed clusters to non-informed, ending when each cluster has received one message. As soon as a cluster receives a message in the second phase (necessarily from an informed cluster), it can broadcast to all its neighbouring non-informed cluster (thereby informing those clusters). The messages being passed are constructed in the same way as in the first phase. When all clusters have received one message, we can extract the scores for any K_i from any cluster containing this subkey using max-marginalization.

Another, similar algorithm combines factors using summation instead of maximization when marginalising variables. This so called sum-product algorithm is more suited for probability queries (e.g. to enable subsequent key enumeration) whereas the max-product algorithm is exact for MAP-queries (and hence optimal for success rate). Out of curiosity, we tried both, and we could hardly see a difference, except for very low success rates where the sum-product algorithm seemed to give a better rank. Since we use success rate as our metric, we stick with the max-product algorithm.

Algorithm 1 Max-Product Algorithm for Full Key Recovery

Input:
Φ a set of factors
$\mathcal{T}$ a clique tree over Φ, with $\mathbf{C}$ its clusters
α initial assignments of factors to clusters
C_r an arbitrarily selected root cluster
Output:
Updated scores $S(K_i = k_i)$ for each K_i

// Initialize $\mathcal{T}$
for each cluster C_i **do**
// Initial factor values of each cluster is the product of their asssigned factors
$\beta_i \leftarrow \prod_{\phi:\ \alpha(\phi)=i} \phi$
for each edge $(i, j) \in \mathcal{T}$ **do**
$\mu_{i,j} \leftarrow 1$ // Setting the initial value of previous message
// Start upward pass
// A cluster is ready when all of its neighbours but one have sent a message
while There is a cluster ready in $\mathbf{C} - C_r$ **do**
Pick i, j such that i is ready to transmit to j
$\delta_{i\to j} \leftarrow \max_{C_i - C_i \cap C_j} \beta_i$
$\beta_j \leftarrow \beta_j \times \frac{\delta_{i\to j}}{\mu_{i,j}}$
$\mu_{i,j} \leftarrow \delta_{i\to j}$
// Start downward pass, C_r is the only informed cluster
while There exist an uninformed cluster **do**
Pick i, j such that i is informed and j is a non-informed neighbour
$\delta_{i\to j} \leftarrow \max_{C_i - C_i \cap C_j} \beta_i$
$\beta_j \leftarrow \beta_j \times \frac{\delta_{i\to j}}{\mu_{i,j}}$
for $i \in [16]$ **do**
Pick any C_j such that $K_i \in C_j$
$S(K_i = k_i) \leftarrow \max_{C_j - K_i} \beta_j$

Cluster Graph Inference Enhanced DPA in Full To conclude, the final workflow of our attack is as follows.

1 **(Compute Distinguishing Scores)** First compute the raw factors for each S-Box by running any distinguisher targeting individual S-boxes, making the attack quite flexible as it allows both non-profiled and profiled attacks (including based on deep learning, see Section 4.4)

2 **(CGI Enhancing)** Use cluster graph inference to combine the factors, creating a unified, global perspective on what are the most likely key byte candidates. One can switch to the min-sum algorithm for combination if the scores are additive rather than multiplicative.

3 **(Recover Full Key)** Use marginalization on the factors to determine the most likely key byte for each subkey and return it. Optionally, perform key enumeration based on the marginalized subkey scores, possibly switching to the sum-product algorithm for the cluster graph inference.

The factor evaluation dominates the complexity of the attack, in particular the ones

with a 16-bit subkey. For every attack trace, 2^{16} values need to be updated, which accounts for most of the work. However, since the S-Boxes have only 256 output values, the amount of pdfs to evaluate does not grow with the subkey.

To give an estimate of the cost of running the max-product algorithm, our non-optimised python implementation (of its min-sum incarnation) on the SKINNY graph runs in approximately ten milliseconds. This runtime is independent of the number of traces, trace acquisition, or distinguisher, as it only combines the scores. Given this low overhead, calculating factors and combining them can be interleaved if need be, retaining the online nature of most existing distinguishers.

4.3 Combining Hi-Dependent Factors (24-bits and Beyond)

So far, we have explained how to combine the bi-dependent factors, thus exploiting the leakage of 44 S-Boxes. We leave the exploitation of all 70 S-Boxes identified in Table 1 as future work, for which there are two challenges to overcome.

The first one is purely computational and relates to factor evaluation. Evaluating scores on 24- and 32-bit subkeys requires a significant amount of computing power. Although, there is precedent for computing these kinds of scores [MOW14, MKP12], it was beyond our resources.

The second one is more technical and relates to the cluster graph inference itself. We mentioned earlier three key bytes were missing from the clique tree. These could be integrated using, for example, the factors $\phi(K_2, K_9, K_{11})$, $\phi(K_2, K_9, K_{12})$ and $\phi(K_1, K_{13}, K_{15})$ from column 3 of Table 1. Adding any more than those three factors of column 3 to the graph would break the running intersection property. Although there do exist (loopy) variants of the max-product algorithm [KF09, 11.3, 13.4], these are neither guaranteed to converge, nor to return the exact result (cf. the use of loopy belief propagation for factor graph inference, see below).

4.4 Cluster Graphs versus Factor Graphs

4.4.1 Large Factor Graph Approximate Inference (LFGAI)

The use of factors and belief propagation to extract keys from leakage is also possible by using factor graphs instead of cluster graphs. In general, factor graphs can represent information in a more fine-grained way than cluster graphs. They are unidirectional bipartite graphs with on the one hand variable nodes and on the other hand factor nodes. A factor node and variable node are connected by an edge if and only if the given factor depends on the respective variable.

Concept Veyrat-Charvillon et al. [VGS14] suggest the use of factor graphs for profiled side-channel analysis as follows. As its variables, they use not only the byte-wise subkeys, but also the byte-wise intermediate values that are leaked upon (plus any additional byte-wise auxiliary values necessary to reach those leaky intermediates). The factors themselves either represent leakage on a given variable, or the relationship between two or three variables according to the cipher's specification (e.g. that one variable is the XOR of two other ones). Notably, for each trace a fresh set of auxiliary and intermediate values is added to the factor graph, thus the full factor graph consists of a central component of common subkey variables to which M identical per-trace subgraphs are connected.

Given a factor graph, belief is propagated along the edges based on two types of messages. Both types encode belief related to the variable connected to that edge, so a single message appears as a score vector (encoding a score for each possible value the variable can take). The messages from variables to factors are simply Hadamard products of the messages (score vectors) received by that variable on the other edges in the previous

iteration on the other edges, whereas the messages from the factors to the variables are of the sum-product type (also based on messages received in the previous round).

The factor graph inference can be initialized by a combination of non-informative priors (for variables without leakage) and initial beliefs for the intermediate values for each trace. The only realistic way of forming these initial beliefs is using some form of profiled distinguisher for the various leaky intermediate, be it a classical template-style distinguisher or a more advanced one, e.g. based on deep learning.

If the factor graph does not contain any loops, then the belief propagation is guaranteed to converge to the exact solution of the inference problem (where the number of steps needed is at most the diameter of the graph). However, if the factor graph does contain loops, the belief propagation becomes loopy and is no longer guaranteed to converge (e.g. it might oscillate) and even if it does, it might only be to an approximate solution [MWJ99]. In the context of side-channel attacks, the relevant large factor graphs are almost always loopy, which makes LFGAI a heuristic approach.

Discussion As mentioned, LFGAI requires profiling of the intermediate variables and it replaces, rather than enhances, how a distinguisher would deal with a large number of traces. As all traces are represented in the large graph and inference can only take place subsequently, LFGAI inherently conflicts with online processing of traces. A related downside of LFGAI are its time and memory complexities, as detailed below (restricting to byte-wise variables).

The main cost in representing the graph itself are the beliefs on the leaky variables (typically stored as a vector of 2^8 floats). When exploiting the same 44 SKINNY S-boxes as we do, LFGAI would lead to a memory complexity of roughly $44 \cdot 256 \cdot M$ floats, i.e. a memory cost linear in the number of traces M.

Calculating the messages for a variable connected to d factors can be done in $d \cdot (d-1) \cdot 2^8$ operations. However, the messages from the factors to the variables are of the sum-product type and, if a factor is connected to d variables, naïvely computing all the messages has complexity $d \cdot 2^{8(d-1)}$. For SKINNY, the factors with $d = 3$ will hence be dominating the computation. These are the XOR operations, whose number in the large factor graph is linear in the number of traces M.

In contrast, CGI can enhance any distinguisher, including non-profiled ones and relies on the distinguisher to process the traces. Thus, for CGI memory complexity is typically independent of the number of traces (as most distinguishers such as CPA and MLE have fixed memory costs), with only the time complexity linear in the number of traces. Moreover, the max-product based post-processing of factors is fast enough that online processing of traces remains an option when the underlying distinguisher supports it.

4.4.2 Independent Factor Graph Approximate Inference (iFGAI)

Green et al. [GRO19] suggested two modifications to factor graph inference. Firstly, in order to reduce the memory complexity, they recommend to create separate, smaller factor graphs for each trace and only combine the beliefs on the key variables across factor graphs towards the end. As this final combination can be done by simply adding or multiplying scores, it can easily be done on-the-fly instead. Thus, this modification allows online processing of traces again and reduces the memory requirements to essentially one small factor graph at a time, making memory consumption independent of the number of traces. Secondly, they propose a method to identify the most informative variables in the factor graph and suggest to remove less informative edges and variables in order to remove any loops and make the factor graph a tree. Although this approach makes the subsequent inference exact, the overall method is still heuristic due to choice of which nodes and edges to remove from the full factor graph. Although Green et al.'s suggestions make factor graph inference more practical, the downside is that the overall attack becomes less potent;

for various of their experiments more traces are needed to reach the same overall success rate as for the large factor graph original.

Discussion Both Veyrat et al. and Green et al. concentrated on AES-128 and identified loops in the factor graph as a potential problem. From that perspective, we can perform a cursory investigation into the likely sources of loops when creating a factor graph to exploit the same leakage for SKINNY as we do.

When considering a factor graph representing only a single trace (as for iFGAI), we observe that the very lightweight features that we exploit, also delays the introduction of loops. Firstly, SKINNY's light key-schedule can be represented in the factor graph as simply variable nodes for each of the key bytes without any further processing. In contrast, for AES, the key schedule itself could introduce loops, e.g. when expressing the relationship between the first and last round keys. Secondly, SKINNY's low-diffusion linear layer can be encoded into a factor graph without inevitably creating loops. In contrast, for AES, already the scalar multiplication by '3' as part of the MixColumns operation created loops in prior work (while that might appear needless, for the overall MixColumns loops do seem inevitable). The consequence for SKINNY is that the creation of loops in the factor graph can be largely be traced back to the same key byte variable contributing to an intermediate in two different ways. Thus, including all the bi-dependent S-boxes will not result in any loops, whereas including all tri-dependent S-boxes definitely will (we did not investigate the maximum number of S-boxes that can be included without incurring loops).

In stark contrast, when considering the large factor graph original, including a single bi-dependent S-box will create a loop as soon as a second trace is included.

Overall, there are indications that factor graph inference will also benefit from SKINNY's lightweight features. Although LFGAI and CGI seem competing ideas with clear qualitative differences (namely LFGAI's dependence on profiling and large memory complexity), we can envision a situation where iFGAI and CGI are complementary, as we explain below. We leave a detailed quantitative comparison an open project (cf. the comparison of LFGAI with algebraic SCA and key enumeration [GS15]). Especially our approach's compatibility with non-profiled attacks raises the fascinating prospect that the earlier factor graph inference might also be mounted in a non-profiled fashion, which we leave as an open problem.

4.4.3 Digging Deeper with Deep Learning

Ever since the introduction of deep learning (DL) to mount side-channel attacks [MPP16], its use has rapidly gained popularity [PPM+23, AGF23]. Against AES-128, a DL-based SCA typically adheres to the classical divide-and-conquer approach, in the sense that each subkey byte is recovered by a separate DL-distinguisher. Furthermore, a DL-distinguisher follows the same two stages as a template attack. During the first stage, traces with known keys and inputs are labelled and used to train a neural network (similar to the profiling stage). In the second stage, target traces with a fixed unknown key are fed to the neural network, receiving scores for each of the possible values for the subkey bytes and scores for a given subkey are accumulated outside the neural network. In our parlance, the output of a given neural network is a factor for the subkey that neural network is trained for; each trace thus leads to a factor and these factors can be combined easily as they are all on the same subkey.

A natural question is to what extent deep learning will be able to exploit deeper structures within a cipher (which is what CGI so neatly achieves). Benadjila et al. [BPS+20, Remark 3] argue that, in the context of AES-128, it makes more sense to train the neural network to learn an intermediate value more closely related to the actual leakage, such as a first round S-box output, rather than the subkey byte that that intermediate leaks upon. The idea is that this change in training target might save the neural network from having

to learn the inverse of the S-box, which can easily be taken care of by an attacker during postprocessing. This perspective indicates that the neural network is mainly exploiting leakage relatively local to a specific intermediate value, as opposed to trying to learn about multiple related intermediate values (e.g. all those related to a single MixColumns column).

Discussion While deep learning is very capable of combining leakage locally, for instance when leakage on a single intermediate variable is spread out due to masking, we are aware of only a few works providing further insight into how neural networks used for SCA might exploit leakage more globally, say by combining S-Box leakage with MixColumns leakage in AES or, more pertinently to our problem, combining leakage from multiple S-boxes.

For instance, Hetwerr et al. [HGG19] used standard explainability techniques from deep learning to attribute the leakage being exploited in a side-channel setting. Here, the application of a saliency map unveiled that a specific attack on an unprotected hardware implementation of AES exploited not only the intended leakage target, but also an additional signal that could subsequently be traced to unintended properties of the hardware implementation of the MixColumns operation. Elsewhere, to test whether a distinguisher uses local or non-local features, Gohr et al. [GJS20] compared the performance of a neural network trained on either real traces or on artificially spliced traces. They found no evidence that their neural network used non-local features, even though an alternative, namely a simple affine extractor on their target, did use non-local features (without those features necessarily being useful).

Finally, when attacking an implementation of Clyde, Gohr et al. [GLS22] observe that their neural network appears to exploit L-box leakage. They express some surprise, as the neural network was trained to predict S-box outputs. From our perspective, the exploited L-box leakage is still relatively local, after all the S-box output is the L-box input; the main challenge was to extract sufficiently useful information from a single trace notwithstanding the masking. Furthermore, to enable the extraction (irrespective of where the leakage emanates), they introduce a simple linear transformation of the labels, dubbed a scattershot encoding; they voice the suspicion that this encoding facilitates the extraction of bits for which a neural network might otherwise face logical difficulties (e.g. when trying to learn a linear function).

In contrast, for our CGI-enhanced DPA against SKINNY, the final marginal distribution of a single subkey depends on the leakage of 44 S-boxes: would a neural network trying to recover a single subkey be able to do the same, would clever labelling of the data help and, if so, would (trained) neural networks for different subkeys hence have a certain overlap or similarity? It appears that, even independent or our work, there are many fascinating open problems left in the domain of DL-SCA beyond divide-and-conquer, especially in scenarios where advanced approaches such as belief propagation are known to outperform simple divide-and-conquer.

Without further investigation, we see DL-SCA as primarily complementary to our work. Indeed, we can even envision a side-channel attack that uses ideas from deep learning, factor graph inference, cluster graph inference, and key enumeration. In the context of SKINNY, such an attack could look roughly as follows.

0 **(Profiling or Training)** For each of the 44 mono- and bi-dependent S-boxes, train a separate neural network to recover the relevant S-box intermediate for any given trace.

1 **(Target Trace Processing)** For each target trace,

   (a) *(Deep Learning)* For each of the S-boxes, run the respective neural network to create beliefs for the intermediate values of that trace.

   (b) *(Factor Graph Consolidation)* Use iFGAI based on the neural networks belief to consolidate a global belief for that single trace.

(c) *(Key factor Update)* Use the iFGAI-consolidated beliefs for the intermediate values to update beliefs for the subkey factors used by CGI, where the update is local (just a simple multiplication or addition).

2 **(CGI Enhancement)** Run CGI to re-consolidate global beliefs about the subkeys using the factors created above.

3 **(Key Enumeration)** Use the marginalized subkey beliefs to enumerate keys. Alternatively, it might be possible to adapt and run a form of list-CGI to enumerate the keys, in a manner similar to list Viterbi or more general N-best algorithms [SC90, NG01].

5 Evaluation of CGI Enhanced DPA

With the theory of CGI enhancement established, we want to determine how effective it is. To this end, we will quantify the security degradation of SKINNY when the extra leakage available to an attacker is combined using CGI. For evaluation purposes, we will stick to the usual metric of the success rate of recovering the key. However, in contrast to the literature, we consider the success rate of recovering the full key, as opposed to the more common success rate of recovering a single key byte only. Our rationale here is that CGI enhancement necessarily involves the full key and even if all S-boxes leak the same (as for the noisy HW scenario), the recovery rates per key byte differ considerably. These differences between various key bytes recovery rates are exacerbated for real traces where S-boxes can leak quite distinctly.

We will compare three different attacks on SKINNY: the naïve divide-and-conquer distinguisher using a single S-Box per key byte, a more evolved version that combines all mono-dependent factors and finally the CGI enhanced attack that exploits all mono- and bi-dependent factors. For the naïve distinguisher, the S-Boxes used are the ones in the top two rows of round 2 and the top two rows of round 40.

We will consider these three attacks in various situations: firstly in the noisy Hamming-Weight (HW) model, and secondly against two real implementations, where we examine both a profiled MLE distinguisher and a non-profiled CPA one. For the HW model we will use noises of $\sigma^2 = 1$ and $\sigma^2 = 4$, respectively, as these values are representative of a low and a high noise regime and roughly match the noise levels we observed for the two real implementations.

5.1 Hamming Weight Model

We start by analysing our attack in the noisy HW model, where initial rounds leak on their S-box outputs and final rounds on their S-box inputs; this model was previously used to compare the leakage of individual lightweight S-boxes [HPGM20, 2.2]. Specifically, a first round S-Box produces leakage ℓ such that:

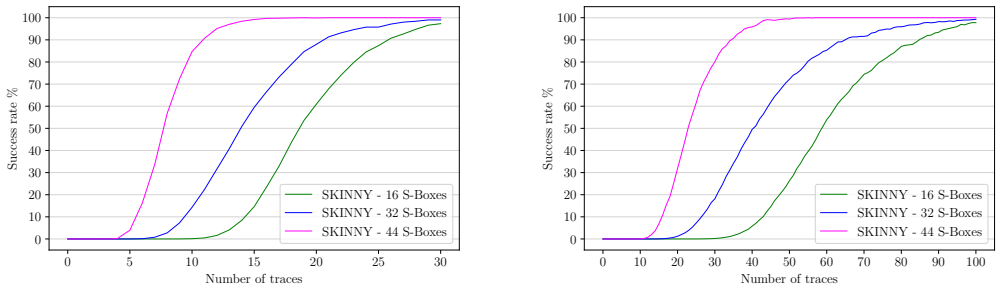
$$\ell = \mathrm{Hw}(\mathsf{SBox}[\mathit{Input}]) + N_{\sigma^2} \,,$$

where N_{σ^2} is sampled from a Gaussian distribution centered on 0 with variance σ^2 and *Input* depends on the plaintext and part of the key. In contrast, a last round S-Box produces leakage ℓ such that:

$$\ell = \mathrm{Hw}(\mathsf{SBox}^{-1}[\mathit{Output}]) + N_{\sigma^2} \,,$$

where *Output* depends on the ciphertext and part of the key.

The results for the three attacks in the noisy HW model are presented in Figure 7, demonstrating that the exploitation of additional factors (32, resp. 44, instead of only 16)



(a) Noisy HW model with $\sigma^2 = 1.0$ **(b)** Noisy HW model with $\sigma^2 = 4.0$

Figure 7: Comparison of success rates of full key recovery on synthetic SKINNY traces exploiting 16 (standard divide-and-conquer), 32 (multi-target, mono-dependent), respectively 44 (CGI-enhanced, bi-dependent) S-boxes using MLE distinguishers for the factors.

provides considerable benefits to an attacker. To get a better impression of the benefits, we can compare the number of traces to reach a given success rate across the attacks (see Figure 1 in the Introduction). It turns out that the benefit of CGI enhancing is slightly more advantageous for the higher noise level ($\sigma^2 = 4$) and appears to decrease ever so slightly for increasing success rates. For instance, for σ^2 we see that the trace reduction granted by CGI enhancing ranges from around 2.7 down to roughly 2.4, slightly below the "theoretical" ratio of number of S-boxes exploited (namely 2.75), yet clearly confirming both the benefits of CGI enhancement and the susceptibility of SKINNY's low diffusion to further SCA exploitation.

5.2 Practical Validation

We established that against SKINNY-128-128 and in the Hamming weight model, the CGI enhanced attack effectively combines the leakage of 44 S-Boxes. We will now show the practicality of the CGI enhanced attack against two real implementations of SKINNY-128-384, which features 56 rounds and a 384-bit tweakey state. This SKINNY variant was used by Romulus in Round 2 of NIST's lightweight standardization process [IKMP20b], although for Round 3, Romulus switched to SKINNY-128-384+ [GIK+21] (which uses only 40 rounds instead of 56; the change should not affect our attack, see also Section 6).

For SKINNY-128-384 as used by Romulus (Round 2), TK_1 and TK_2 are used for the tweak and TK_3 contains the key. We collected traces on a ChipWhisperer platform, targeting a Cortex-M4 running two different implementations of SKINNY-128-384, one based on lookup tables (LUT) and another circuit-based one. We used the same device for profiling and attacking, which intuitively gives a worst-case scenario for the defender [BCH+20]. For both implementations, we collected 50 000 profiling traces with varying key and plaintext for every trace, and 50 000 attack traces with a single fixed key and randomly varying plaintexts for every trace. Tweakeys TK_1 and TK_2 were set to fixed known values, distinct for the profiling and attack traces. The attack traces were split in 500 experiments of 100 attack traces each for the circuit implementation and 1000 experiments of 50 attack traces each for the LUT one.

Both implementations, integrated with the ChipWhisperer code, are available at https://github.com/Simula-UiB/CGI-DPA, along with the code used to run the experiments.

5.2.1 Experimental Setup

Targeted Implementations We target two software implementations of SKINNY, both written in ARM assembly. In both implementations, the round keys are precomputed and stored row-wise in blocks of 32 bits. Similarly, the cipher state is stored row-wise in 4 registers, so each register holds an entire state row. Using assembly code ensures that the state does not leave those registers during the computation, thus avoiding any unwanted `load` or `store` operations. As SKINNY's linear layer and tweakey schedule boil down to a couple of `xor`, `rot` and round key `load` operations, the only difference between the two implementations is the S-Box computation.

The first implementation comes directly from the FELICS framework [DBG+15] and uses a precomputed lookup table to evaluate the S-Boxes. It computes each S-Box using the following assembly snippet:

```
and  r7, r2, #0xff  //selecting the first byte of the state
ldrb r7, [r6,r7]    //loading the corresponding SBOX output in r7
bfi  r2, r7, #0, #8 //inserting back in the state the result value
```

where `r7` is a temporary usage register, `r2` points to a row of the state and `r6` points to the address of the S-Box lookup table. As the `load` is performed on the output of the S-Box, we expect the implementation to match the Hamming weight model (with high SNR).

In the second implementation, we replaced the lookup table with a circuit implementation, casting the S-Box as an efficient boolean circuit using eight `NOR` and eight `XOR` operations (taken from the SKINNY specification).

Since we store four bytes of the state per register, we can compute this circuit in a SIMD fashion on an entire row. The exact implementation of this circuit was directly taken from skinny-c [Wea18], compiled using the `-O3` flag and inserted manually into the assembly. In contrast to the first implementation, this one avoids `load` and `store` operations on the S-Box outputs; moreover, it computes four S-Box simultaneously, thus reducing the SNR by introducing algorithmic noise. However, the circuit evaluation takes longer, giving rise to more points of interest.

For a simple comparison between the two implementations, we can look at an SNR plot (Figure 8). As expected, the LUT implementation leaks much more and during a shorter time window, with a peak SNR of around 2.3, which roughly matches $\sigma^2 = 1.0$ in the HW model. The circuit version leaks across multiple points with a peak SNR of approximately 0.35, comparable to the HW model with $\sigma^2 = 4.0$.

Trace Acquisition We collected traces for both implementations on an STM32F303, which uses a Cortex-M4. More specifically, we used the ChipWhisperer platform and its onboard power measurement setup. During the acquisition, the MCU ran at 7.37MHz, and the ADC ran at 4×7.73MHz, sampling four timestamps per clock cycle. As our entry level ChipWhisperer can only capture 5000 timestamps per trace, we recorded the first few and last few rounds separately for every input, merging corresponding front and end traces in post-processing. All our recorded traces are available online (see https://github.com/Simula-UiB/CGI-DPA's `README`).

Template Attacks We first exploit those traces with a template attack [CRR03]. A template is a multivariate Gaussian distribution that characterizes power traces for a particular intermediate variable. We use the 50 000 profiling traces to build templates for each of the 44 targeted S-Boxes, following the standard two steps of point selection and subsequent estimation of multivariate Gaussian distributions [MOP07]. The attack then uses template matching to obtain our factors. Subsequently, we combine these factors using the max-product algorithm as explained in Section 4,

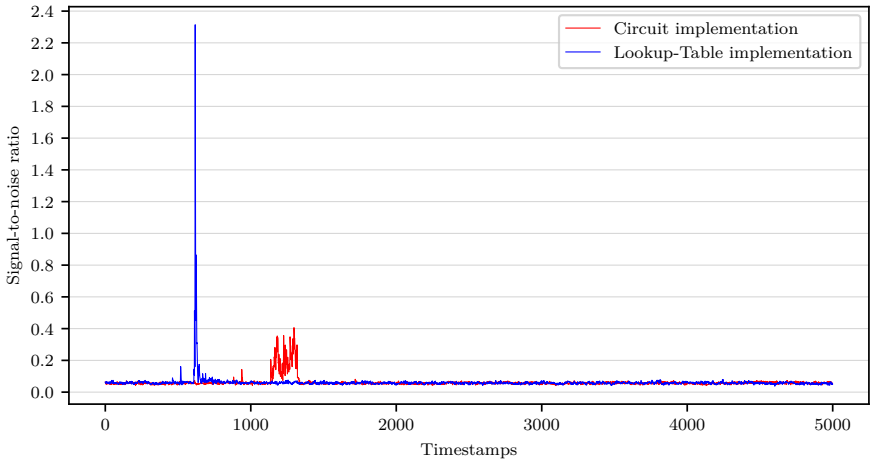
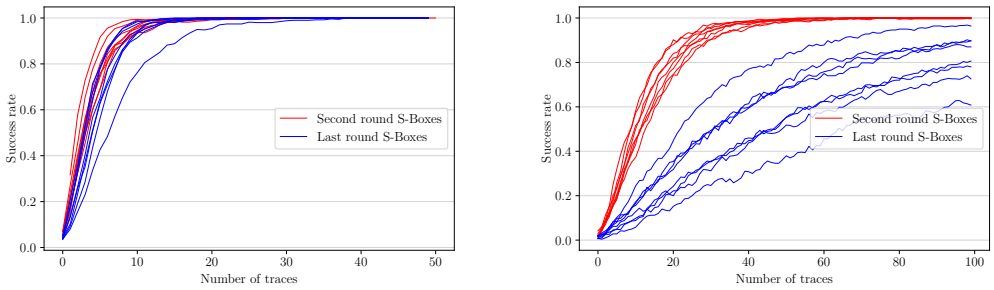


Figure 8: SNR of a 1st round S-Box on a Cortex-M4 for two SKINNY implementations



(a) Success rate for LUT implementation

(b) Success rate for circuit implementation

Figure 9: Success rate of individual S-Boxes on SKINNY on the simple template attack

For point selection we opted for the *Sum of Difference* method [RO05]. It selects points of interest (PoIs) by partitioning the traces by the value of a targeted intermediate variable, computing the mean power consumption for every partition and summing the pairwise difference between those means. The timestamps which maximize the sum indicate the most interesting PoIs. For this approach, the number of PoI is a parameter, that we set to 4 for both implementations (more points requires more profiling traces).

For the Gaussian distributions, we estimated a separate covariance matrix for each possible value of the targeted intermediate, leading to 256 covariance matrices for each S-Box (we did not encounter numerical stability issues).

Correlation Power Analysis We also exploit our traces using the unprofiled Pearson correlation coefficient distinguisher [BCO04] to demonstrate the robustness of our attack. We only use the attacking traces this time, as no profiling is required. Contrary to template attacks, we also do not select PoIs. However, we define a "window of interest" of 400 timestamps for each S-Box. This windowing was made using SNR computations but could be done by visually inspecting the traces.

We attack each S-Box using the standard Correlation Power Analysis (CPA): we assume that the power consumption follows the HW model and compute the correlation between this hypothetical power model and each timestamp in our window of interest. After which,

the score for each subkey value is the maximum correlation value obtained across all timestamps. We then plug those scores into CGI.

As correlation-based scores are additive [LPR+14, Proposition 2], we use the additive version of the Max-Product algorithm, incorporating all the necessary changes (such as initialization to 0 instead of 1 etc.).

5.2.2 Results

Profiling Distinguisher Figure 10 provides the success rates as a function of the number of traces when running the three different attacks (exploiting 16, 32, resp. 44 S-boxes) on either the LUT (left) or circuit (right) implementation. We use the same comparison as previously with the noisy HW model. For the LUT implementation, our CGI-enhanced attack reduces the number of traces required to break SKINNY by a factor of 2.75 (Figure 1), matching the noisy HW model.

The circuit implementation has even higher efficacy, between 5 and 6. This efficacy is due to the success rate of the attack using 16 S-Boxes being unreasonably low compared to what we would expect. While it is not shown in Figure 10, as all the S-Boxes are combined, it is the last round S-Boxes which are responsible for that low success rate (individual success rates for each S-Box are available in Figure 9). While we know that last-round S-Boxes tend to have a lower success rate [HPGM20], this discrepancy is more than what we see in the LUT implementation and thus probably stems directly from the leakage.

We reckon that the circuit implementation leaks primarily on the output of the S-Box computation and not much on the input. This type of leakage can create a situation where attacking a single S-Box gives leakage that is linearly related to the ciphertext and makes it hard to recover the key, whereas combining two different S-Boxes can surpass this problem. It speaks in favour of our attack, showing that it can combine S-Boxes despite their poor success rate.

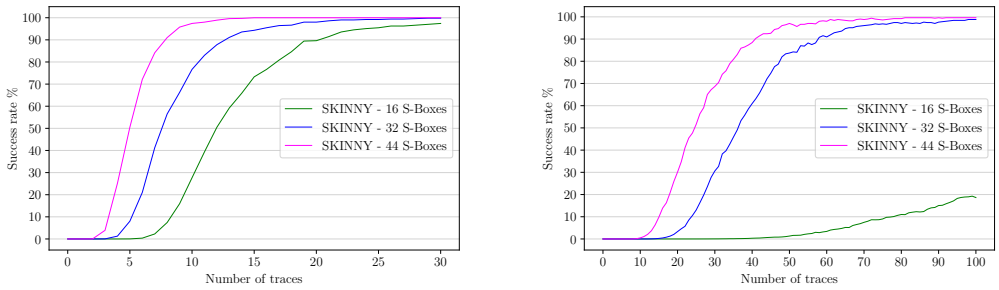
Non-Profiling Distinguisher Using the same set of traces as for the template attacks, we present the results of using an unprofiled CPA distinguisher in Figure 11. As CPA tends to be less potent than a template attack, the number of traces available was too low for the baseline CPA attack to recovering the full key (recall that we report the success rate of recovering all key bytes, not just one). The CPA attack that uses all mono-factors performs slightly better, particularly for the circuit version. However, CGI-enhanced CPA does recover the entire key effectively, with 60% and 85% success rates for the two respective implementations.

Those results demonstrate the potency of our attack compared to some of the other advanced attacks we discussed earlier in Section 4.4. It does not require profiling and hence, allows the attacker to choose their best distinguisher available without constraints. In conclusion, our attack is practical and effectively exploits SKINNY's low diffusion.

6 Conclusion, Impact and Mitigations

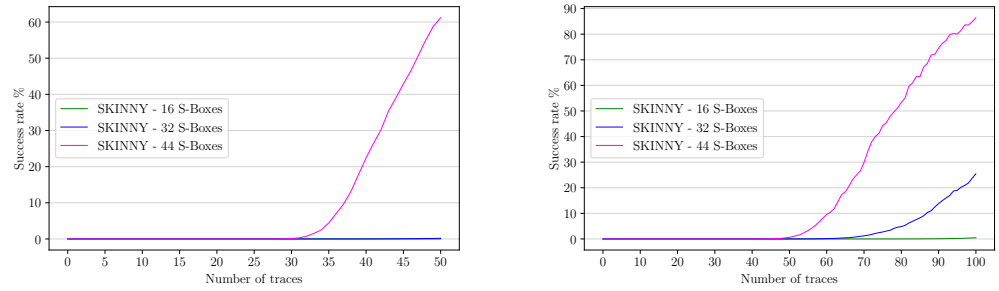
6.1 Applying CGI-DPA to Different Variants of SKINNY

The original SKINNY paper describes six different SKINNY variants, to which a seventh (SKINNY-128-384+) was later added. The differences between the variants are the block size (a nibble-oriented 64-bit or a byte-oriented 128-bit), the size of the tweakey state (1, 2 or 3 times the block size) and the number of rounds (32, 36, 40, 48 or 56). When the tweakey size is 2 or 3 times the block size, the tweakey state is divided into blocks denoted TK_i. During the tweakey addition, the top two rows of each TK_i are XORed to the top two rows of the state. Subsequently, each block is updated using P_T (as described in Section 3), and an LFSR is applied to each byte of TK_2 and TK_3.



(a) LUT implementation **(b)** Circuit implementation

Figure 10: Comparison of success rates of full key recovery on real SKINNY traces exploiting 16 (standard divide-and-conquer), 32 (multi-target, mono-dependent), respectively 44 (CGI-enhanced, bi-dependent) S-boxes using profiled MLE distinguishers for the factors.



(a) LUT implementation **(b)** Circuit implementation

Figure 11: Comparison of success rates of full key recovery on real SKINNY traces exploiting 16 (standard divide-and-conquer), 32 (multi-target, mono-dependent), respectively 44 (CGI-enhanced, bi-dependent) S-boxes using CPA distinguishers for the factors.

For the byte-oriented variants of SKINNY, adding multiple TK_i allows for different key layouts and lengths. All known schemes using SKINNY keep the 128-bit key laid out in a single TK_i (TK_3 for Romulus). In that case, our initial analysis of SKINNY-128-128 still applies. The extra LFSRs of TK_2 and TK_3 apply to each byte individually: the bytes in each TK_i still do not mix.

The number of rounds can impact our attack, as it affects the number of iterations of the key schedule permutation P_T before the last rounds and, by extension, the clique tree for the cluster graph inference. Fortunately, as we briefly touched on in Section 3, P_T has a period of 16, so the clique trees for 40 rounds (SKINNY-128-128 or SKINNY-128-384+) and 56 rounds (SKINNY-128-384) are identical. For SKINNY-128-256 (48 rounds), the factors obtained in the penultimate round differ and produce a different clique tree.

For the nibble-oriented variants, the 128-bit key is spread over multiple tweakey blocks, presenting an extra set of challenges. On the one hand, 32-bit subkeys now correspond to 4 key nibbles, so hi-dependent S-boxes become enumerable, yet on the other hand effectively and efficiently combining the factors to reconstruct the full key is still an open problem.

6.2 CGI-DPA in the Presence of Countermeasures

We attacked unprotected implementations in the Hamming weight model and on an ARM Cortex-M4. As we saw for the CPA examples against the ARM implementations, as

long as there exists a distinguisher targeting individual S-Boxes and producing scores for the underlying subkeys, these scores—or factors—can be combined using CGI. More realistic implementations will be protected by countermeasures, such as masking. For masking, higher order DPA can still target individual S-boxes, resulting in scores that we can interpret as factors, and our attack still applies, in the sense that we can exploit the extra S-boxes.

Of course, when attacking a masked implementation, we would expect to need more traces and the per-trace cost evaluation of the factors will likely go up, yet the cost of subsequent combination using CGI will not. We expect that the same rule of thumb will still apply: that the increase in the number of exploitable S-boxes leads to a commensurate reduction in the number of traces needed. We leave verification of this conjecture an open problem.

6.3 Slow Diffusion Considered Harmful?

SKINNY is not the only lightweight cipher to feature the new paradigm of slow diffusion with many rounds. We mentioned LED, which allows for a much simpler pincering [HPGM20], but also GIFT [BPP+17], a blockcipher for another round-three candidate [BCI+21], and ASCON [DEMS21], a keyed sponge that was recently selected by NIST for lightweight standardization, appear susceptible to CGI-enhanced DPA. Both share some CGI-friendly features with SKINNY: a slow diffusion with a very slim permutation layer and a very straightforward relationship between the first and last round keys. However, both schemes differ from SKINNY as they are far more bit-oriented, which could complicate CGI.

GIFT features a Present-style bit-level permutation and nibble-size S-Boxes. Furthermore, the key is added at bit-level, with a key schedule such that any given bit of any round key depends only on a single bit of the master key. Consequently, factors will be over selections of key bits, dispensing the neat byte-structure of the factors that we found so useful for SKINNY. When considering GIFT-128, the first round S-boxes do not consume any key material; the second round S-boxes consume half of the key with each (nibble) S-box depending on two key-bits; and the third round S-boxes each depend on two bits of the second half of the key plus, through the state, on eight bits of the first half of the key. For the final rounds the situation is reversed, in the sense that the last round S-boxes depend on two bits of the second half of the key and the penultimate round S-boxes on two bits of the first half plus eight from the second half. While these (128) S-boxes are all eminently enumerable, the resulting cluster graph will be considerably more complicated and—we did not check—unlikely to be a clique tree, thus combination of factors might have to rely on loopy belief propagation instead. For GIFT-64, only a fourth of the master key is consumed per round, and as a consequence the same four rounds only leak on half of the key once and on the other half five times and again, (loopy) CGI appears the ideal tool to combine all this leakage.

Similarly, for ASCON, the key is injected, unchanged, both during the initialization and the finalization of the encryption process. The knowledge of the nonce can be used to mount an attack in the first round of the initialization [SD17] while the knowledge of the tag allows to mount an attack on the last round of the finalization [YKSH22]. Like GIFT, combining the two sides might require a more complicated (loopy?) cluster graph but again we leave it as an open problem.

Our results highlight a tension in lightweight design. A key scheduler that completely lacks diffusion allows an SCA attacker to combine leakage from the first and last rounds. Furthermore, combined with slow diffusion within the cipher's state, it increases the number of S-Boxes that only depend on a small number of key bits (say fewer than 32), likely resulting in more exploitable leakage.

On the other hand, while slow diffusion does not provide a defensive advantage, a lighter key schedule makes for cheaper re-keying potentially creating a trade-off. As

re-keying more frequently diminishes the amount of traces an attacker can use to mount an attack, it might outweight the gains of the better analytical attack enabled by the lighter key-scheduler. We leave as an open problem the systematic comparison of the cost and benefits of different key-scheduling strategies in the context of belief propagation based side-channel attacks, such as CGI-enhanced DPA. Furthermore, we believe that slow diffusion within the cipher state should be avoided when possible as it creates issues when SCAs are taken into account. While the key-scheduler gave us an initial edge on SKINNY, it was the slow diffusion that we leveraged the most in our attack.

Acknowledgement

We thank the anonymous reviewers for their feedback, including useful pointers to related work. We are grateful to Thomas Marquet for insightful discussions related to the potential of deep learning for side-channel attacks.

References

[ABB+20] Melissa Azouaoui, Davide Bellizia, Ileana Buhan, Nicolas Debande, Sébastien Duval, Christophe Giraud, Éliane Jaulmes, François Koeune, Elisabeth Oswald, François-Xavier Standaert, and Carolyn Whitnall. A systematic appraisal of side channel evaluation strategies. In Thyla van der Merwe, Chris J. Mitchell, and Maryam Mehrnezhad, editors, *SSR 2020*, volume 12529 of *LNCS*, pages 46–66. Springer, Heidelberg, 2020.

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, November 2001.

[AGF23] Rabin Yu Acharya, Fatemeh Ganji, and Domenic Forte. Information theory-based evolution of neural networks for side-channel analysis. *IACR TCHES*, 2023(1):401–437, 2023.

[BB17] Paul Bottinelli and Joppe W. Bos. Computational aspects of correlation power analysis. *Journal of Cryptographic Engineering*, 7(3):167–181, September 2017.

[BBC+20] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 369–400. Springer, Heidelberg, August 2020.

[BCH+20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *NDSS 2020*. The Internet Society, 01 2020.

[BCI+21] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, and Siang Meng Simand Yosuke Todo. GIFT-COFB v1.1. National Institute of Standards and Technology (NIST), Lightweight Cryptography Standardization, 2021.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors,

CHES 2004, volume 3156 of *LNCS*, pages 16–29. Springer, Heidelberg, August 2004.

[BDM+20] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 311–341. Springer, Heidelberg, May 2020.

[BGL09] Lejla Batina, Benedikt Gierlichs, and Kerstin Lemke-Rust. Differential cluster analysis. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 112–127. Springer, Heidelberg, September 2009.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. Cryptology ePrint Archive, Report 2016/660, 2016. https://eprint.iacr.org/2016/660.

[BJK+20] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. SKINNY-AEAD and SKINNY-Hash. *IACR Transactions on Symmetric Cryptology (S1)*, pages 88–131, 2020.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, Heidelberg, September 2007.

[BOW15] Valentina Banciu, Elisabeth Oswald, and Carolyn Whitnall. Exploring the resilience of some lightweight ciphers against profiled single trace attacks. In Stefan Mangard and Axel Y. Poschmann, editors, *COSADE 2015*, volume 9064 of *LNCS*, pages 51–63. Springer, Heidelberg, 2015.

[BPP+17] Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, Heidelberg, September 2017.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, June 2020.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Heidelberg, August 2003.

[DBG+15] Dumitru-Daniel Dinu, Alex Biryukov, Johann Groszschädl, Dmitry Khovratovich, Yann Le Corre, and Léo Paul Perrin. FELICS - fair evaluation of lightweight cryptographic systems. National Institute of Standards and Technology (NIST), Workshop on Lightweight Cryptography, 2015.

[De 20] Lauren De Meyer. Looking at the NIST lightweight candidates from a masking point-of-view. Cryptology ePrint Archive, Report 2020/699, 2020. https://eprint.iacr.org/2020/699.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. National Institute of Standards and Technology (NIST), Lightweight Cryptography Standardization, 2021.

[Fei70] Hörst Feistel. Cryptographic coding for data-bank privacy. IBM Technical Report RC-2827, 1970.

[GIK+21] Chun Guo, Tetsu Iwata, Mustafa Khairallah, Kazukiho Minematsu, and Thomas Peyrin. Romulus, v1.3. National Institute of Standards and Technology (NIST), Lightweight Cryptography Standardization, 2021.

[GJS20] Aron Gohr, Sven Jacob, and Werner Schindler. Subsampling and knowledge distillation on adversarial examples: New techniques for deep learning based side channel evaluations. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 567–592. Springer, Heidelberg, October 2020.

[GLS22] Aron Gohr, Friederike Laus, and Werner Schindler. Breaking masked implementations of the clyde-cipher by means of side-channel analysis A report on the CHES challenge side-channel contest 2020. *IACR TCHES*, 2022(4):397–437, 2022.

[GNS05] P. J. Green, Richard Noad, and Nigel P. Smart. Further hidden Markov model cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 61–74. Springer, Heidelberg, August / September 2005.

[GPPR12] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw. The LED block cipher. Cryptology ePrint Archive, Report 2012/600, 2012. https://eprint.iacr.org/2012/600.

[GRO19] Joey Green, Arnab Roy, and Elisabeth Oswald. A systematic study of the impact of graphical models on inference-based attacks on AES. In Begül Bilgin and Jean-Bernard Fischer, editors, *CARDIS 2018*, volume 11389 of *LNCS*, pages 18–34. Springer, Heidelberg, 2019.

[GS15] Vincent Grosso and François-Xavier Standaert. ASCA, SASCA and DPA with enumeration: Which one beats the other and when? In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 291–312. Springer, Heidelberg, November / December 2015.

[HGG19] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 645–666. Springer, Heidelberg, August 2019.

[HPGM20] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Lightweight ciphers and their side-channel resilience. *IEEE Transactions on Computers*, 69:1434–1448, 2020.

[HRG14] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. Good is not good enough - deriving optimal distinguishers from communication theory. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 55–74. Springer, Heidelberg, September 2014.

[IKMP20a] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Duel of the titans: The Romulus and Remus families of lightweight AEAD algorithms. *IACR Trans. Symm. Cryptol.*, 2020(1):43–120, 2020.

[IKMP20b] Tetsu Iwata, Mustafa Khairallah, Kazukiho Minematsu, and Thomas Peyrin. Romulus, v1.2. National Institute of Standards and Technology (NIST), Lightweight Cryptography Standardization, 2020.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 274–288. Springer, Heidelberg, December 2014.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

[KR11] Lars R. Knudsen and Matthew J.B. Robshaw. *The Block Cipher Companion*. Information Security and Cryptography. Springer, Heidelberg, Heidelberg, 2011.

[KW03] Chris Karlof and David Wagner. Hidden Markov model cryptanalysis. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 17–34. Springer, Heidelberg, September 2003.

[LPdH10] Jiqiang Lu, Jing Pan, and Jerry den Hartog. Principles on the security of AES against first and second-order differential power analysis. In Jianying Zhou and Moti Yung, editors, *ACNS 10*, volume 6123 of *LNCS*, pages 168–185. Springer, Heidelberg, June 2010.

[LPR+14] Victor Lomné, Emmanuel Prouff, Matthieu Rivain, Thomas Roche, and Adrian Thillard. How to estimate the success rate of higher-order side-channel attacks. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 35–54. Springer, Heidelberg, September 2014.

[Man03] Stefan Mangard. A simple power-analysis (spa) attack on implementations of the aes key expansion. In Pil Joong Lee and Chae Hoon Lim, editors, *ICISC 2002*, volume 2587 of *LNCS*, pages 343–358. Springer, Heidelberg, 2003.

[MKP12] Amir Moradi, Markus Kasper, and Christof Paar. Black-box side-channel attacks highlight the importance of countermeasures - an analysis of the xilinx virtex-4 and virtex-5 bitstream encryption mechanism. In Orr Dunkelman, editor, *CT-RSA 2012*, volume 7178 of *LNCS*, pages 1–18. Springer, Heidelberg, February / March 2012.

[MMOS16] 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. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 548–572. Springer, Heidelberg, December 2016.

[MOOS15] Daniel P. Martin, Jonathan F. O'Connell, Elisabeth Oswald, and Martijn Stam. Counting keys in parallel after a side channel attack. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 313–337. Springer, Heidelberg, November / December 2015.

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

[MOW14] Luke Mather, Elisabeth Oswald, and Carolyn Whitnall. Multi-target DPA attacks: Pushing DPA beyond the limits of a desktop computer. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 243–261. Springer, Heidelberg, December 2014.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *SPACE 2016*, volume 10076 of *LNCS*, pages 3–26. Springer, Heidelberg, 2016.

[MWJ99] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belie, berlin, germany, april 13-14, 2015. revised selected papersf propagation for approximate inference: An empirical study. In Kathryn B. Laskey and Henri Prade, editors, *UAI'99*, pages 467–475. Morgan Kaufmann, 1999.

[NG01] Dennis Nilsson and Jacob Goldberger. Sequentially finding the N-best list in hidden Markov models. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, volume 2 of *IJCAI'01*, page 1280–1285, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[NIS18] Submission requirements and evaluation criteria for the lightweight cryptography standardization process. National Institute of Standards and Technology (NIST), 2018.

[NIS22] Heavyweight security via lightweight cryptography. Meltem Sonmez Turan, National Institute of Standards and Technology (NIST), Invited talk at Real World Crypto, January 2022.

[Osw03] Elisabeth Oswald. Enhancing simple power-analysis attacks on elliptic curve cryptosystems. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 82–97. Springer, Heidelberg, August 2003.

[PGA+23] Kostas Papagiannopoulos, Ognjen Glamočanin, Melissa Azouaoui, Dorian Ros, Francesco Regazzoni, and Mirjana Stojilović. The side-channel metrics cheat sheet. *ACM Comput. Surv.*, 55(10), February 2023.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 513–533. Springer, Heidelberg, September 2017.

[PPM+23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. SoK: Deep learning-based physical side-channel analysis. *ACM Comput. Surv.*, 55(11):1–35, February 2023.

[RO05] Christian Rechberger and Elisabeth Oswald. Practical template attacks. In Chae Hoon Lim and Moti Yung, editors, *WISA 2004*, volume 3325 of *LNCS*, pages 440–456. Springer, Heidelberg, 2005.

[RS10] Mathieu Renauld and François-Xavier Standaert. Algebraic side-channel attacks. In Feng Bao, Moti Yung, Dongdai Lin, and Jiwu Jing, editors, *Information Security and Cryptology (Inscrypt 2009)*, volume 6151 of *LNCS*, pages 393–410. Springer, Heidelberg, 2010.

[SC90] R. Schwartz and Y.-L. Chow. The N-best algorithms: an efficient and exact procedure for finding the N most likely sentence hypotheses. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 81–84, 1990.

[SD17] Niels Samwel and Joan Daemen. Dpa on hardware implementations of ascon and keyak. In *Proceedings of the Computing Frontiers Conference*, page 415–424, New York, NY, USA, 2017. Association for Computing Machinery.

[Sha49] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

[SWT01] Dawn Xiaodong Song, David A. Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In Dan S. Wallach, editor, *USENIX Security 2001*. USENIX Association, August 2001.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 282–296. Springer, Heidelberg, December 2014.

[Wea18] Rhys Weatherley. skinny-c. https://github.com/rweather/skinny-c, 2018.

[YKSH22] Shih-Chun You, Markus G. Kuhn, Sumanta Sarkar, and Feng Hao. A template attack on ascon aead. CHES 2022 - Poster, September 2022.

[YMO22] Rebecca Young, Luke Mather, and Elisabeth Oswald. Comparing key rank estimation methods. In Ileana Buhan and Tobias Schneider, editors, *CARDIS'22*, volume 13820 of *LNCS*, pages 188–204. Springer, Heidelberg, 2022.