A Comparison of Graph-Inference Side-Channel Attacks Against SKINNY

Stian Husum ✉, Håvard Raddum, and Martijn Stam

Simula UiB, Bergen, Norway.
{stianh,haavardr,martijn}@simula.no

Abstract. Belief propagation can improve standard divide-and-conquer side-channel attacks by exploiting additional leakage both effectively and efficiently. The original approach for belief propagation against block ciphers uses factor graph inference (FGI). Recently, Costes and Stam (CHES'23) proposed the use of cluster graph inference (CGI) as a more effective and potentially more efficient alternative. In the context of SKINNY, they focus on exploiting leakage on the 44 S-boxes that each depend on at most two subkeys, surprisingly enabling exact inference. Expanding their cluster graph approach beyond the 44 S-boxes is intrinsically expensive, as it requires enumerating larger subkeys.
In contrast to CGI, FGI remains efficient regardless of the number of S-boxes exploited, yet in practice exploiting more rounds appears to quickly yield diminishing returns: against AES so far only two rounds have been exploited effectively. Costes and Stam provided a rough, qualitative comparison between cluster and factor graph inference, but without any quantitative experiments. Thus, it remains unclear how well FGI would fare against a low-diffusion cipher like SKINNY.
We provide a quantitative comparison of the two graph inference methods applied to SKINNY. We conclude that, when profiling is possible, both behave comparably when exploiting the aforementioned 44 S-boxes. Yet, FGI can easily exploit more S-boxes, comprehensively outperforming CGI. For the profiled scenarios originally considered by Costes and Stam, FGI on three leaking rounds from both sides of the cipher is best, both in terms of effectiveness and efficiency.

Keywords: Belief Propagation · Graph Inference · SKINNY

1 Introduction

Several approaches have been proposed to squeeze more information out of side-channel analysis (SCA) leakage than the typical divide-and-conquer strategies (D&C). In D&C the target key is attacked by dividing it into several subkeys. Each of those subkeys are attacked separately by applying a distinguisher to a set of attack traces, yielding a score for each possible subkey. The most likely subkey is then determined from the highest score. However, a full trace usually contains leakage in later rounds that relies on the master key in more complicated ways, which cannot be exploited by standard D&C.

The use of probabilistic graphical modeling and belief propagation algorithms has been proposed to exploit more of the observed leakage [5, 24]. Two approaches, based on factor graphs and cluster graphs, respectively, look particularly promising, yet it is unclear how they relate to each other.

Veyrat-Charvillon et al. [24] advocated for factor graphs to model interactions between dependent variables in the cipher, followed by factor graph inference (FGI) to recover the full key. At its core, FGI builds a bipartite graph with nodes representing variables, respectively factors. For all nodes, local beliefs encode how likely the variables relevant to that node are deemed to be. Typically, for a select number of variable nodes, informative initial beliefs are available and subsequent belief propagation allows consolidating these partial, initial beliefs into a consistent, global belief on all the variables encoded by the graph.

In the context of SCA, the variables relate to the intermediate variables in the cipher, such as subkeys or S-box inputs and outputs, whereas the factors capture the operations on those variables, such as xors and S-boxes. Moreover, a typical SCA factor graph consists of a component encoding the key schedule and a component for the evaluation of the cipher proper. To combine information from multiple traces, we can use large FGI (LFGI), in which the cipher component is replicated for each trace and connected by a shared key schedule component. For the intermediate variables that leak directly during a computation, the initial beliefs correspond to the distinguisher scores for that particular intermediate (for intermediates that do not leak directly, say subkeys, a uniform prior can be used as initial belief). As beliefs are needed for intermediates appearing in a single trace, the distinguisher will have to be parameterized to obtain meaningful initial beliefs, which in turn requires a profiling stage to estimate the parameters involved (e.g. for Gaussian templates). Following the belief propagation, the final beliefs for the subkeys reveal the best guess for the full key.

A downside of FGI in SCA are the large and loopy graphs. Consequently, the belief propagation is only approximate and might not even converge. In principle, FGI-SCA could encode the full cipher and thus initial beliefs on all S-box inputs and outputs could be entered. Yet, when targeting a byte-wise implementation of AES, it turns out that leakage beyond the first round only provides marginal benefits [6, 9] and the benefit of FGI over D&C lies in the former's ability to exploit leakage from the first round's MixColumns.

Recently, Costes and Stam [5], henceforth CS23, introduced cluster graph inference (CGI) to SCA. Like factor graphs, cluster graphs are known from probabilistic graphical modeling, but they differ as they only have one type of nodes (for variables) and edges can be drawn between nodes to indicate shared variables. In the context of SCA, each node in the cluster graph contains a set of subkeys, and each possible value for the subkeys in that set must be enumerated for the attack. Thus, a node corresponding to two subkey bytes corresponds to an enumeration effort of 2^{16} values. This enumeration scales poorly with deeper rounds in a cipher where the leakage depends on more than two subkey bytes.

CS23 applied CGI to the lightweight cipher SKINNY. Due to SKINNY's slower diffusion, CS23 observed that 44 of its S-boxes each depend on at most two

subkey bytes. The resulting cluster graph happens to be a clique tree, enabling exact inference. For this case, CS23 demonstrated that both in a profiled and unprofiled setting, the number of traces needed is roughly inversely proportional to the number of S-boxes exploited. So, using CGI against SKINNY's 44 bi-dependent S-boxes is $\approx 2.75 = 44/16$ times as efficient as exploiting only a single S-box per subkey in standard D&C.

When comparing CGI and FGI, attacking AES with FGI can exploit all rounds efficiently in principle, but in reality not effectively, as almost all useful leakage occurs in the first round [6]. After two full rounds of AES, every byte of the cipher state depends on the entire key. Conversely, with SKINNY's slower diffusion, in the third round some bytes in the cipher state still only depend on two keybytes. When attacking SKINNY with CGI, CS23 leverage this slow diffusion and effectively exploit deeper rounds in the cipher, but they cannot go deeper than three rounds efficiently. An open question is how well FGI might exploit slow diffusion by targeting variables deeper in a cipher and how well it compares to CGI, especially in scenarios where the latter still provides exact inference.

Our Contribution. We will compare CGI and FGI, specifically LFGI, using the lightweight blockcipher SKINNY as example, thus enabling a fair comparison with CS23's results (on CGI). We construct factor graphs for SKINNY, and use them to mount FGI attacks on synthetic traces with Gaussian noise, $\sigma^2 = 1$ and $\sigma^2 = 4$, as well as traces from two real implementations of SKINNY: one with a LUT implementation for the S-box and another with a circuit-based one representative of a scenario with low noise, respectively high noise. We compare those FGI attacks to the CGI attacks performed by CS23. For our comparison, we look at the success rates of the attacks over the number of traces and restrict ourselves to the profiled setting as, unlike CGI, FGI only works with profiled distinguishers.

We show that when FGI and CGI are exploiting the same leakage from the same 44 S-boxes, both attacks reach a similar success rate with the same amount of traces. However, unlike CGI, FGI can handle a larger amount of target S-boxes, and we demonstrate that additional S-box leakage leads to attacks with higher success rates. We targeted up to 128 S-boxes for the same scenarios as CS23, with a significant reduction in the traces needed for an attack. In addition, we investigated FGI in a synthetic scenario with even higher noise, $\sigma^2 = 16$, and saw that going beyond 128 S-boxes did not yield better results. Unlike FGI attacks on AES, where mainly the first round leads to leakage, FGI on SKINNY can exploit leakages from four rounds on each side of the cipher, leading to eight rounds, or 128 S-boxes, of leakage. Although FGI does exploit more S-boxes, the reduction in traces does not follow the linear relation suggested by CS23, with 96 S-boxes having a better reduction than the expected sixfold, and further rounds having a worse reduction than expected.

Related Works. Veyrat-Charvillon et al. [24] introduced FGI to target AES, and several papers since have studied and revised the method [6, 7, 9, 23]. In addition, it has been employed to attack other targets than AES, such as Keccak [12, 25], ASCON [14, 26], and Clyde [2], as well as the number-theoretic transform, or NTT, used in lattice-based cryptography [19, 20]

Several strategies have been suggested for scaling FGI for multiple traces, mainly large FGI (LFGI) and independent FGI (IFGI). The strategy of LFGI was employed in the initial attack by Veyrat-Charvillon et al. [24] and duplicates the graph for each trace and connects them through the key schedule. IFGI, on the other hand, introduced by Green et al. [6], uses a separate graph for each trace, and then consolidates the shared beliefs of the graphs after belief propagation. The main benefit of IFGI is improved efficiency with larger amounts of traces without significantly sacrificing the quality of the inference. As the trace count for our attacks are rather low, our focus will be on LFGI.

Prior to FGI, ASCA [21, 22] and similar attacks [16, 17] were suggested to exploit leakage in the traces beyond standard D&C. These attacks rely on an algebraic set of equations generated from the cipher, and augment that system with side-channel leakage before solving. The solvers used by ASCA use hard values and so cannot handle the noisy measurements from SCA directly. Strategies have been suggested to bridge the gap between solver and leakage in ASCA [18], but probabilistic belief propagation methods need no such bridge. Grosso and Standaert [7] compared FGI with ASCA arguing in favour of FGI, yet Strieder et al. [23] demonstrated an attack on the AES key schedule where ASCA outperforms FGI. We focus on FGI and leave algebraic side-channel attacks on SKINNY as open problem.

2 Preliminaries

2.1 SKINNY

SKINNY [1] is a family of lightweight tweakable blockciphers. A variant of it was a final-round candidate in the NIST lightweight cipher standardization [8]. It is constructed as a substitution-permutation network, similar to AES, but with slower diffusion, a lighter key schedule, and no key whitening. As SKINNY follows the tweakey framework [11], it uses in a tweakey instead of a key. A part of that tweakey is the secret key, and the rest is a public tweak.

Within the family, there are two different block sizes, 64 and 128 bits, as well as three different tweakey sizes, n, $2n$, and $3n$ bits, where n is the block size in bits. We describe each variant using these two values, so SKINNY-128-384 has a block size of 128 bits and a tweakey size of 384 bits ($3n$). The cipher state is divided into a 4×4 grid of cells, each cell holding either a nibble or a byte, depending on the size of the block. We index each cell in a block from 0 to 15 as a subscript. So, the top left cell of a state x is x_0 and the bottom right is x_{15}. The number of rounds depends on both the block and key size. The variants we will consider, SKINNY-128-128 and SKINNY-128-384, have 40 and 56 rounds,

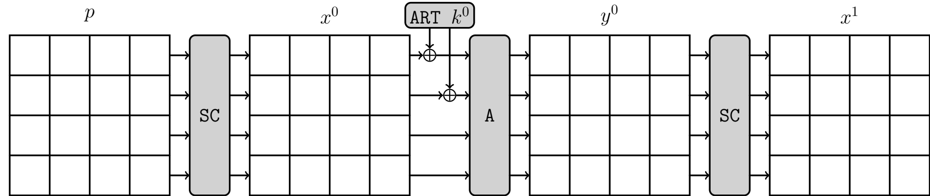

Fig. 1: The inital rounds of SKINNY. Adapted from [10].

respectively. Below we provide a high level overview of SKINNY; for full details please refer to the SKINNY specification [1]. We refer to the beginning rounds of the cipher, i.e., from the plaintext side, as the *initial* rounds, and to the ending rounds, i.e., from the ciphertext side, as the *final* rounds. The first round (and a bit) is illustrated in Fig. 1, starting with the plaintext p.

Round Structure and Round Functions. A round starts with applying an S-box function to each cell. We refer to this layer as the S-box layer (`SC`), and the S-box as S. After the S-box layer, the round tweakey is added to the upper half of the state (`ART`). After the tweakey addition, there is an affine layer consisting of adding round constants, shifting rows, and mixing columns. For simplicity, We depict the full affine layer as `A`.

As the first S-box layer comes before the first key addition, it contains no key material, and we can compute it without access to the key. Therefore, we choose to index rounds starting at the first S-box output as x^0. We denote the output of the affine layer, i.e., the end of a round, and input to the next S-box layer, as y^r. Consequently, the first S-box input with key material is y^0. Following this notion, the ciphertext is y^{39} or y^{55}, depending on the variant.

Key Schedule. The SKINNY key schedule is lightweight. First, the tweakey is divided into several blocks, each being a 4×4 grid of cells. Each block has the same size as the state, and the size of the tweakey denotes the number of blocks. So, SKINNY-128-128 has one block, and SKINNY-128-384 has three blocks. The secret key is stored in the last block of the tweakey; earlier blocks contain the public tweak. Between rounds, the blocks are updated independently of each other, by applying both a permutation to shuffle a block's cells and a bitwise-linear substitution on each of the block's cells. The permutation

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

is the same for all blocks. This permutation is structured such that the two halves of the key are separate, so the first half of the key feeds into even rounds and the second half into odd rounds.

On the other hand, the linear substitution depends on the block. For the first block, and only block in SKINNY-128-128, it is the identity map (and can hence

be omitted). For the two other blocks, present in SKINNY-128-384, a linear feedback shift register (LFSR) is applied to each cell in the upper half of the key state after the permutation. This LFSR is different for the two blocks, but since the last block is the one containing the secret key in SKINNY-128-384, we choose to denote that LFSR as L.

2.2 Side-Channel Analysis

In power analysis, the attacker measures a *trace* of the power drawn by a device during encryption to try to recover the secret key on the device. The trace contains multiple measurements per clock cycle, and therefore contains the power drawn by the operations in the cipher. We consider the plaintext and ciphertext to be known to the attacker, as is the cipher specification and the implementation running on the device.

The basis for most attacks is targeting some intermediate variables used during the computation of the cipher. A *distinguisher* is used on the traces to determine which value is the most likely for a given variable. The distinguisher is a function that takes a trace and outputs scores, a non-negative real number, for each possible value the intermediate variable can have. For variables that do not change between executions of the cipher, like subkeys and variables in the key schedule, a distinguisher might take several traces.

In traditional divide-and-conquer SCA attacks, the target key is divided into subkeys that are attacked independently. Such attacks are simple and efficient, however, they do not exploit points in the trace where the leakage is joint between several subkeys. As such, they do not exploit all available leakage in the traces. To remedy this, several approaches to combine the information from multiple intermediate variables have been proposed [5, 21, 22, 24]. The details of two of those, FGI and CGI, will be described in Sections 2.3 and 3.

2.3 Probabilistic Graphical Models

We can look at our side-channel leakage as a probabilistic system, with the leakage of each intermediate variable as a random variable. As the number of random variables in our system grows, the complexity of the distribution increases, and we need an efficient way to represent the system. Probabilistic graphical modeling provides one way of accomplishing this by using a graph-based representation of joint distributions. The graph representation allows us to run *inference* effectively on the system. We use uppercase letters to denote random variables in our systems. Our description of probabilistic graphical modeling is based on Koller and Friedman [13]. Further details can be found in their book.

A central concept in probabilistic graphical modeling is that of a *factor*. A factor is a mapping from the value space of some random variables to a non-negative real number. We use the Greek letter ϕ to denote factors. The set of random variables included in a factor are called the *scope*, so a factor $\phi(K_i, K_j)$ has scope K_i, K_j. When we evaluate a factor, we use lower case instantiations

of the random variables, so an instantiation of the random variable K_i would be k_i.

We can look at distinguishing scores from SCA as factors. We use multiplicative notation for factors, so if we want to combine factors we multiply them like we would with probabilities. However, in our implementation we work additively, by taking the logarithm, for better numerical stability.

Factors as Nodes. Factors form the main basis for our graph representation. In cluster graphs, each node is a *cluster* of factors, i.e., each node represents a subset of all factors in our system. We then draw edges between nodes that share factors.

In factor graphs, we have a bipartite graph with two types of nodes: factors and variables. We represent every factor and every variable by a node. We draw edges from each factor node to the variable nodes in its scope. Our figures show factor nodes with boxes and variable nodes without.

Belief Propagation. We employ *belief propagation* to run inference on our graphical models. A *belief* is a real non-negative number associated with the possible values a variable can have, where a higher number indicates a stronger belief that a variable takes a particular value. As such, a factor can be described as a set of beliefs for its scope. In each node, we store a belief on the variables relating to that node, with the Greek letter β denoting that belief:

– A cluster node stores beliefs on the variables in the scope of its factors.
– A variable node stores beliefs on its variable.
– A factor node stores beliefs on its neighbouring variables, where beliefs on different variables are independent from each other.

After initializing some nodes in a graph with beliefs given by the traces, with unitialized nodes having uniform beliefs, we propagate those beliefs through the graph by *message passing*. Each node sends beliefs along its edges and then uses the beliefs it receives to update its own beliefs. We repeat sending messages, receiving messages, and updating beliefs until the overall beliefs in the graph are consolidated.

Non-loopy graphs consolidate their beliefs after a fixed number of iterations and yield an exact inference. However, if the graphs have loops, we must define a termination condition, such as a fixed number of iterations. With loopy graphs, the beliefs might not consolidate, and we only get an *approximate* inference.

Cluster Graph Inference. CS23 takes the cluster graph approach, and take the key byte variables as factors. They select each S-box dependent on exactly two key bytes and declare those pairs of key bytes as clusters. For SKINNY, this yields 12 clusters containing 13 of the 16 key bytes. Those clusters serve as the nodes for their cluster graph. Edges are drawn between the clusters to form a clique tree. The clique tree used they used for SKINNY-128-128 can be

found in Figure 6 in CS23. The key bytes not included in the graph are attacked separately with a standard divide-and-conquer approach.

To populate the inital beliefs for the nodes, CS23 uses either a profiled or unprofiled distinguisher to collect scores on keybytes and pairs of keybytes. Those scores are the factors. Each node is associated with three factors $\phi_1(K_i, K_j)$, $\phi_2(K_i)$ and $\phi_3(K_j)$. Then, for all $k_i, k_j \in \{0,1\}^8$, the belief $\beta(k_i, k_j)$ is initialized to $\phi_1(k_i, k_j) \cdot \phi_2(k_i) \cdot \phi_3(k_j)$.

After the beliefs on the nodes have been initialized, they run two passes of belief propagation: first from leaf nodes up to an arbitrary root node, then from the root node and back to the leaves. Once beliefs have been consolidated, they can be extracted from the nodes by *max-marginalization*, i.e., for a node K_i, K_j, the belief on $\beta(k_i) = \max_{k_j} \beta(k_i, k_j)$.

The nodes in the cluster graph contain factors relating to two key bytes, so computing the scores for those factors require a computational cost on the order of 2^{16}. Expanding the attack to more S-boxes, and therefore S-boxes dependent on three or more key bytes, would increase the cost to 2^{24} or beyond. For this reason CGI gets prohibitively expensive at deeper rounds.

On the other hand, only targeting the key bytes allows for a larger diversity of distinguishers, and therefore CGI can be used in both profiled and unprofiled scenarios. In addition, as the CGI graph for SKINNY-128-128 and SKINNY-128-384 using 44 S-boxes neatly forms an acyclic graph, we know that the inference we have is exact, and the attacker does not have to tweak parameters such as the number of message passing iterations.

3 Large Factor Graph Inference

Factor Graph Inference for SCA. Recall that factor graphs are bipartite graphs consisting of factor nodes and variable nodes. When applied to SCA, the variable nodes represent intermediate variables in the computation of the cipher, and the factor nodes correspond to operations in the cipher. This approach was first introduced by Veyrat-Charvillon et al. [24] to attack AES.

The graphs consist of two main components: the key schedule, whose values do not change between traces, and the cipher proper. For each target trace, we replicate the cipher component and connect them to the shared key schedule component. Green et al. [6] refer to this method of connected components as large FGI (LFGI, as opposed to independent or sequential FGI). They deem that method preferable when graphs remain small and will therefore be our approach.

We compute initial beliefs from the traces using a distiguisher and use those beliefs to initialize the variable nodes. Each variable node correspond to one intermediate variable, so if our variables are bytes, they store 2^8 beliefs. Hence, the size of the nodes does not increase the deeper we go into the cipher. However, to get beliefs on deeper intermediate variables, we need to use a distinguisher whose complexity also does not grow with depth. Therefore, FGI is better suited for use with profiled distinguishers.

After initializing the beliefs in the variable nodes, we propagate those beliefs using message passing. If the graph has no cycles propagation takes a fixed number of steps, yielding an exact inference. When the graph contains loops, we continue until either some convergence criteria is met, or after some heuristically set number of iterations. For loopy graphs, FGI only yields an approximate inference. After belief propagation, we can extract beliefs on subkeys from the variable nodes relating to the subkey variables.

3.1 Factor Graphs for SKINNY

To attack SKINNY using FGI, we first need a factor graph to describe the operations of the cipher. This graph can either be specific to one implementation, and hence generated from it, as in Veyrat-Charvillon et al. [24], or, as we choose to do, described manually from the cipher specification. Although our approach omits some potential leakage from real implementations, it provides a fairer comparison with CS23 and serves as the worst case for FGI (from an adversarial perspective). As we will be performing attacks on both SKINNY-128-128 and SKINNY-128-384, we need two constructions for our graphs. Both graph variants share the same cipher component, but differ in the key schedule.

For the cipher component we only need two types of factor nodes, S-boxes and XORs. The S-box nodes define a relationship between nodes x_i^{r+1} and y_i^r as

$$x_i^{r+1} = S[y_i^r]$$

where i is the index of the byte, r is the round number and S is the SKINNY S-box. For the XOR nodes, each node defines an XOR between several inputs and one output. An XOR node f_n has a set of neighbours V and defines the relation

$$\bigoplus_{v \in V} v = 0.$$

Fig. 2 shows one round of the cipher component of the graph. We repeat this partial graph for each initial and final round we attack, and then the combined graph is replicated for each of our traces. Finally, we connect each of the graphs through the key schedule. Although we can construct a small loopless graph for slightly more than one round, such a graph becomes loopy once several subgraphs are connected through the key schedule. Thus, all the graphs we construct are loopy and the subsequent inference is heuristic.

For SKINNY-128-128, the key schedule is only a permutation. Therefore, we only need one set of key variable nodes k_n for $n \in \{0, ..., 15\}$ and k_n^r is only a renaming of that variable, i.e., $k_n^0 = k_n$ and

$$k_n^r = k_{P[n]}^{r-1}$$

where P is the key schedule permutation. As the first round of SKINNY contains no key material, we chose to exclude that round from the graph and start at the first S-box output.

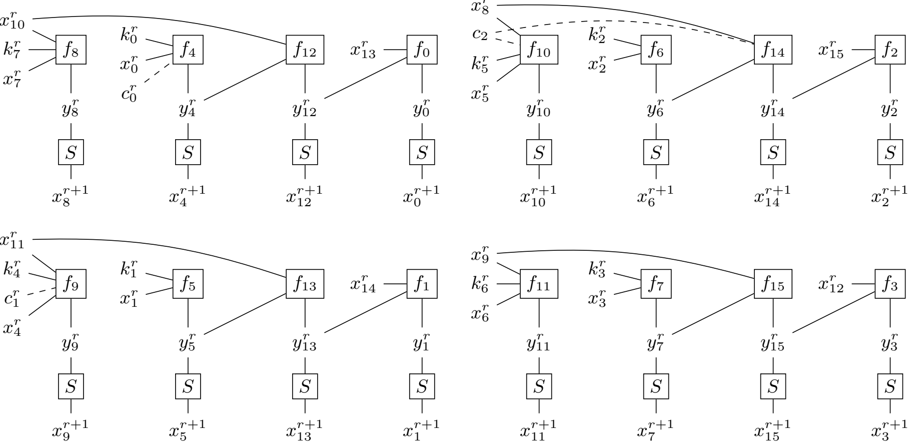

Fig. 2: One round of the FGI graph used for SKINNY. Square nodes are factors, and the rest are variables. Dashed lines indicate known values. Those are included in the factor calculation, but are not variable nodes in the graph for belief propagation.

For SKINNY-128-384, the key schedule contains an LFSR, so an additional factor node is needed. A node L defines, similar to the S-box nodes, a relation between two variable nodes k_n and $k^r_{P^r[n]}$ as

$$k^r_n = L^{\left\lfloor \frac{r+1}{2} \right\rfloor}[k_{P^r[n]}]$$

where $L^{\left\lfloor \frac{r+1}{2} \right\rfloor}$ indicates $\left\lfloor \frac{r+1}{2} \right\rfloor$ applications of the SKINNY LFSR and P^r is r applications of the key schedule permutation. As the first round key is half of the initial key, we have $k^0_n = k_n$ for $n \in \{0, ..., 7\}$.

As SKINNY only uses half the key in each round, we only need to use eight key variable nodes for each of our rounds. We also add the known tweak to each XOR node that is already neighbour with a key variable.

We also constructed an alternative factor graph, where we connect k^i to k^{i-2} through a single invocation of L, instead of directly connecting k^i to k^0 through $L^{\left\lfloor \frac{r+1}{2} \right\rfloor}$. This approach increases the diameter of the graph and makes the path from each round key to the master key less direct. Especially the final rounds end up further from the master key. When we tried message passing on this alternative indirect graph it performed worse when attacking more than 96 S-boxes and performed the same for fewer S-boxes.

4 Experiments and Results

For our comparison between CGI and FGI, we consider the profiled scenarios from CS23: a simulated scenario targeting SKINNY-128-128 with noisy Ham-

ming weight leakage from S-boxes, and a profiled attack on two implementations of SKINNY-128-384 collected on a ChipWhisperer. As there is no clear way to use FGI for an unprofiled attack, we do not compare against CS23's CPA attacks.

The results for CGI are taken directly from data provided to us by the authors of CS23, as are the traces used for our attacks on real traces. As CS23 we use the first order success rate of full key recovery as our metric for comparison.

Following CS23, we look at the leakage of S-box outputs of the initial rounds of encryption, and S-box inputs for the final rounds of the encryption.

4.1 FGI Setup

For the FGI attack, we use the SCALib library [3]. This library provides a full implementation of the FGI algorithm with access to APIs to: Create factor graphs from a text description, duplicate parts of those graphs to accommodate multiple traces, set known values, add initial beliefs to variables, run exact belief propagation on acyclic graphs, run approximate belief propagation on cyclic graphs with a fixed number of iterations, and reading out beliefs of variables. Therefore, the choices we have available are the structure of the graph, which nodes to add initial beliefs to, what those beliefs are, which belief propagation strategy to use, and the number of iterations.

For our attacks we generate a text representation of the graphs in Section 3.1 to describe the factor graphs. This representation skips the first S-box layer, as that layer does not contain any key material, so the known plaintexts must go through one S-box layer before being passed to SCALib. We chose to only look at S-box leakages, and therefore we only set initial beliefs for S-box input and/or outputs. Since the graphs are cyclic, we run approximate belief propagation. As we will be doing LFGI attacks, we use SCALib's built-in support for expanding the graphs. The code used for our experiments is available at `https://github.com/Simula-UiB/SKINNY-LFGI` and the trace dataset has been uploaded [4].

As noted earlier, the number of iterations of message-passing for approximate inference to consolidate, assuming it does, depend on the diameter of the graph. Going from one trace to two traces doubles the diameter, however, increasing from two to three or more makes no change. Therefore, the iteration count is independent of the number of traces. On the other hand, changing the number of rounds *does* change the diameter, and therefore impacts the iteration count. By taking the largest graph we look at, SKINNY-128-384 with 7 rounds on both sides, and increasing the iteration count of the attack until the increase in success rate started to diminish, we determined 60 iterations to be sufficient for our comparison. The success rates for FGI attacks on that graph with 25 target synthetic traces having a noise of $\sigma^2 = 16$ can be seen in Fig. 3. The noise was chosen such that the attack would not have a too high success rate at too few traces, and 25 target traces was chosen to not hit 100% success rate.

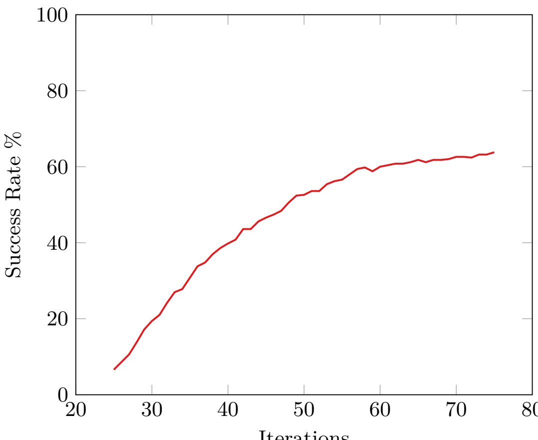

Fig. 3: Success rate as a function of number of iterations in the belief propagation step. Graph constructed as FGI targeting 224 S-boxes (7+7 rounds) of SKINNY-128-384 with 25 synthetic traces of noise $\sigma^2 = 16$. Success rate computed from 500 experiments.

4.2 Synthetic Traces with Gaussian Noise

We start in the synthetic scenario, with leakage modeled as Hamming weights with Gaussian noise. As we know the parameter of the noise, we assume perfect templates and use those for the attack. That is, for each intermediate variable v, the leakage is sampled from $\mathcal{N}(\mathrm{HW}(v), \sigma^2)$. Let $f(x|\mu, \sigma^2)$ be the p.d.f. for $\mathcal{N}(\mu, \sigma^2)$, the normal distribution of mean μ and variance σ^2. Then the initial beliefs β for a node in the factor graph is computed as

$$\beta(i) = f(l|\mathrm{HW}(i), \sigma^2) \tag{1}$$

for $0 \leq i \leq 255$.

CS23 only consider the leakage of S-boxes. With FGI, we could target all intermediate values, but for a more direct comparison, we chose to do the same as CS23, attacking only S-boxes. In each of the attacks, we look at the S-box outputs of initial rounds, and S-box inputs of final rounds. In addition, they only look at the 44 bi-dependent S-boxes, so for our initial comparison we will first look at an attack with only the same S-boxes to form our baseline. On the same scenarios as them, we then extend our attack to target up to four rounds from both sides, leading to a total of 128 S-boxes.

For each experiment, when another trace is added, the previous traces stay the same. So the experiments for n traces is the same as the experiment for $n-1$ traces, but with one additional trace added. For each plot, the experiments are repeated 500 times with new random traces.

Fig. 4 shows the results for these scenarios for $\sigma^2 = 1$ and $\sigma^2 = 4$. We can see that for 44 S-boxes, the two approaches are comparable. However, FGI is

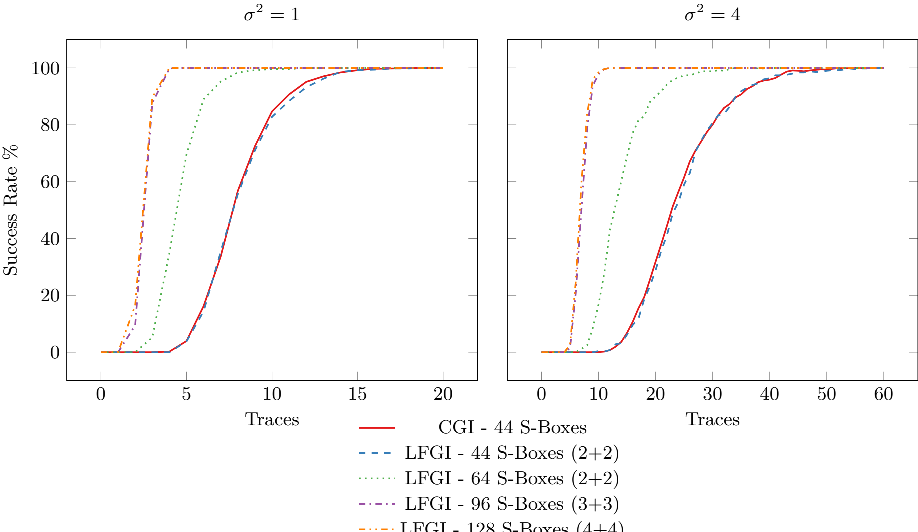

Fig. 4: Full key success rates over number of traces for FGI and CGI attacking generated Hamming weight traces of SKINNY. For the plaintext rounds, the S-box outputs are used, and for the ciphertext rounds, the S-box inputs are used.

able to exploit the information in the added S-boxes, and we can see that those additional S-boxes yield enough information for FGI to have a higher success rate with fewer traces.

Compared to a D&C approach, a fourfold increase in target S-boxes, from 16 to 64, yield a fourfold reduction in traces needed to reach similar success rates. The same linear relation between success rate and the number of S-boxes used approximately holds also when the number of target S-boxes is 44. However, for 96 S-boxes the reduction in traces needed is greater than sixfold reduction expected, and for 128 S-boxes there is no significant improvement over 96 S-boxes. Therefore, this relation does not seem to hold for deeper rounds.

Both $\sigma^2 = 1$ and $\sigma^2 = 4$ yield attacks with high success rates at only a handful of traces. Therefore, it is hard to see if more rounds would benefit the attack. To compare more rounds, we also attacked synthetic traces with higher noise, as seen in Fig. 5. Here, the noise level is $\sigma^2 = 16$ for synthetic SKINNY-128-128 traces, with 500 experiments for each number of traces. In addition to the success rates, we have computed estimated key ranks using SCALib and provide the average of the base two logarithm of those key ranks, following the suggestion of Martin et al. [15]. We can see from both success rates and average key ranks that there is an improvement going from 96 to 128 S-boxes, but going deeper is not beneficial for our approach. As the success rate shows a clearer separation between the different methods, we will stick to success rate henceforth.

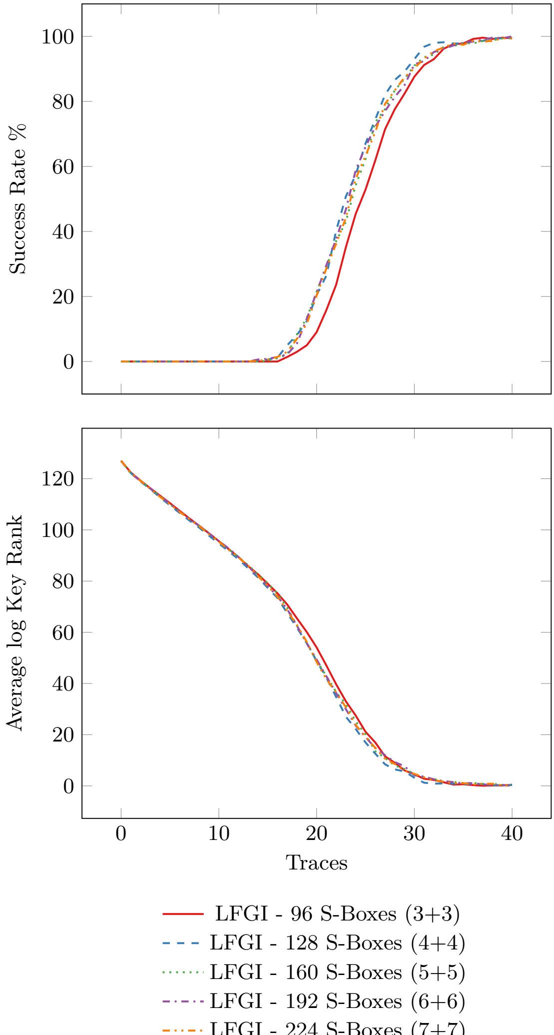

Fig. 5: Full key success rates and average $\log_2$ key ranks over number of traces for FGI attacking generated Hamming weight traces of SKINNY-128-128. For the plaintext rounds, the S-box outputs are used, and for the ciphertext rounds, the S-box inputs are used. Noise is $\sigma^2 = 16$.

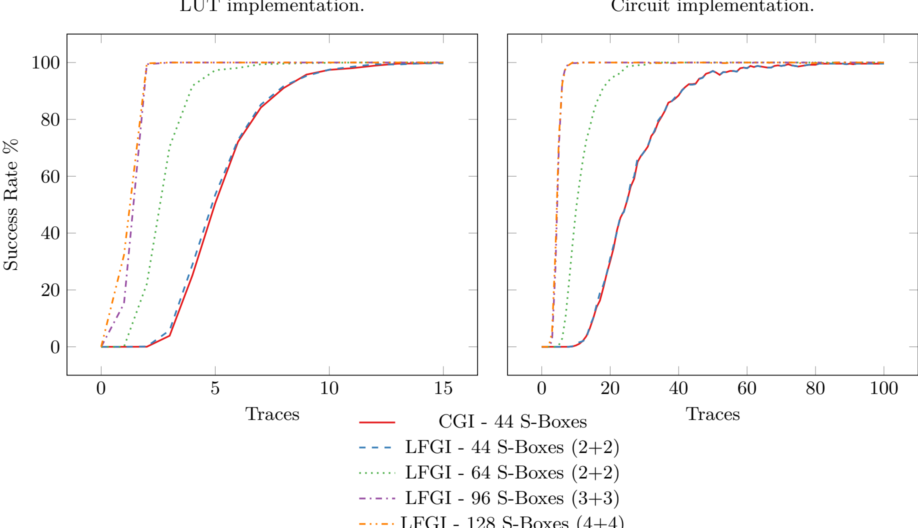

Fig. 6: Full key success rates over number of traces for FGI and CGI attacking SKINNY traces gathered on a ChipWhisperer. For both attacks, points of interest are found using sum of differences and scores are computed using Gaussian templates.

4.3 Real traces produced by ChipWhisperer

For the real traces, we consider two implementations. One implements S-boxes as lookup tables (LUT) and the other with a circuit. Both of these implementations are of SKINNY-128-384, with the target key stored in the last block of the tweakkey. As the LUT leaks more, it will represent a scenario with low noise. As the circuit leaks less, it will be representative of a high-noise scenario. Since SKINNY-128-384 uses LFSRs in the key schedule, the graphs for FGI need to change to include them, as described in Section 3.1.

For our profiled FGI attack we create profiles in the same way as CS23. For each of our target S-boxes, we select points of interest (PoI) using sum of differences, then we compute multivariate Gaussian templates for all 256 possible values of the S-boxes at those PoI.

For the attacks, we compute initial beliefs using our profiled templates on the attack traces. For the number of experiments we follow CS23 and do 500 experiments for the attacks on the circuit implementation and 1000 experiments for the attacks on the LUT implementation.

We can see from Fig. 6 that FGI is, again, on par with CGI with an equal amount of S-boxes, but FGI is able to exploit deeper rounds. Attacking 64 S-boxes yields an improved attack, which is further improved with 96 S-boxes, but there is no significant difference when going from 96 to 128 S-boxes.

5 Conclusion

We demonstrated an LFGI attack on SKINNY by constructing factor graphs for SKINNY-128-128 and SKINNY-128-384. We compared the performance of the attacks in a profiled setting with synthetic and real traces. Our attacks outperformed the previous state-of-the-art CGI attack by CS23, with the LFGI attacks exploiting leakage deeper into the cipher. However, unlike CGI, LFGI is not applicable in an unprofiled setting.

Our comparison focused on the graphical inference step in the attacks, as such, other aspects are left to optimize for a full attack. Our attack could be combined with deep learning based attacks, or improved dimensionality reduction techniques such as LDA. We also did not study the efficacy of our attack against masked implementations, so that remains a topic for future research.

For our experiments, we used success rate as our metric to compare the different attacks. An alternative approach would be comparing keyranks. However, we also believe key enumeration strategies is a potential point of improvement for graph based attacks, and leave that for future research.

Several open questions also remain for optimizing FGI for SCA. What is a good heuristic for the iteration count for loopy graphs? How do we quantify the impact of loops in our graphs, and what is a sensible heuristic for removing loops? How do we use FGI in an unprofiled setting?

References

1. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 123–153. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53008-5_5
2. Bronchain, O., Standaert, F.X.: Breaking masked implementations with many shares on 32-bit software platforms. IACR TCHES **2021**(3), 202–234 (2021). https://doi.org/10.46586/tches.v2021.i3.202-234, https://tches.iacr.org/index.php/TCHES/article/view/8973
3. Cassiers, G., Bronchain, O.: Scalib: A side-channel analysis library. Journal of Open Source Software **8**(86), 5196 (2023). https://doi.org/10.21105/joss.05196, https://doi.org/10.21105/joss.05196
4. Costes, N., Husum, S., Stam, M., Raddum, H.: Side-channel analysis traces for SKINNY (Jan 2025). https://doi.org/10.5281/zenodo.14640175
5. Costes, N., Stam, M.: Pincering SKINNY by exploiting slow diffusion enhancing differential power analysis with cluster graph inference. IACR TCHES **2023**(4), 460–492 (2023). https://doi.org/10.46586/tches.v2023.i4.460-492
6. Green, J., Roy, A., Oswald, E.: A systematic study of the impact of graphical models on inference-based attacks on AES. In: Bilgin, B., Fischer, J. (eds.) CARDIS'18. LNCS, vol. 11389, pp. 18–34. Springer, Heidelberg (2019). https://doi.org/10.1007/978-3-030-15462-2_2
7. Grosso, V., Standaert, F.X.: ASCA, SASCA and DPA with enumeration: Which one beats the other and when? In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015,

Part II. LNCS, vol. 9453, pp. 291–312. Springer, Heidelberg (Nov / Dec 2015). https://doi.org/10.1007/978-3-662-48800-3_12
8. Guo, C., Iwata, T., Khairallah, M., Minematsu, K., Peyrin, T.: Romulus v1.3 specification (2021)
9. Guo, Q., Grosso, V., Standaert, F.X., Bronchain, O.: Modeling soft analytical side-channel attacks from a coding theory viewpoint. IACR TCHES **2020**(4), 209–238 (2020). https://doi.org/10.13154/tches.v2020.i4.209-238, https://tches.iacr.org/index.php/TCHES/article/view/8682
10. Jean, J.: TikZ for Cryptographers. https://www.iacr.org/authors/tikz/ (2016)
11. Jean, J., Nikolic, I., Peyrin, T.: Tweaks and keys for block ciphers: The TWEAKEY framework. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 274–288. Springer, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45608-8_15
12. Kannwischer, M.J., Pessl, P., Primas, R.: Single-trace attacks on keccak. Cryptology ePrint Archive, Report 2020/371 (2020), https://eprint.iacr.org/2020/371
13. Koller, D., Friedman, N.: Probabilistic Graphical Models - Principles and Techniques. MIT Press (2009)
14. Luo, S., Wu, W., Li, Y., Zhang, R., Liu, Z.: An efficient soft analytical side-channel attack on ascon. In: Wang, L., Segal, M., Chen, J., Qiu, T. (eds.) WASA'22. LNCS, vol. 13471, pp. 389–400. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-19208-1_32
15. Martin, D.P., Mather, L., Oswald, E., Stam, M.: Characterisation and estimation of the key rank distribution in the context of side channel evaluations. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 548–572. Springer, Heidelberg (Dec 2016). https://doi.org/10.1007/978-3-662-53887-6_20
16. Oren, Y., Kirschbaum, M., Popp, T., Wool, A.: Algebraic side-channel analysis in the presence of errors. In: Mangard, S., Standaert, F.X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 428–442. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-15031-9_29
17. Oren, Y., Renauld, M., Standaert, F.X., Wool, A.: Algebraic side-channel attacks beyond the hamming weight leakage model. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 140–154. Springer, Heidelberg (Sep 2012). https://doi.org/10.1007/978-3-642-33027-8_9
18. Oren, Y., Wool, A.: Tolerant algebraic side-channel analysis of AES. Cryptology ePrint Archive, Report 2012/092 (2012), https://eprint.iacr.org/2012/092
19. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. In: Schwabe, P., Thériault, N. (eds.) LATINCRYPT 2019. LNCS, vol. 11774, pp. 130–149. Springer, Heidelberg (Oct 2019). https://doi.org/10.1007/978-3-030-30530-7_7
20. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 513–533. Springer, Heidelberg (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4_25
21. Renauld, M., Standaert, F.X.: Algebraic side-channel attacks. Cryptology ePrint Archive, Report 2009/279 (2009), https://eprint.iacr.org/2009/279
22. Renauld, M., Standaert, F.X., Veyrat-Charvillon, N.: Algebraic side-channel attacks on the AES: Why time also matters in DPA. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 97–111. Springer, Heidelberg (Sep 2009). https://doi.org/10.1007/978-3-642-04138-9_8

23. Strieder, E., Ilg, M., Heyszl, J., Unterstein, F., Streit, S.: ASCA vs. SASCA - A closer look at the AES key schedule. In: Kavun, E.B., Pehl, M. (eds.) COSADE 2023. LNCS, vol. 13979, pp. 65–85. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-29497-6_4
24. Veyrat-Charvillon, N., Gérard, B., Standaert, F.X.: Soft analytical side-channel attacks. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 282–296. Springer, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45611-8_15
25. You, S., Kuhn, M.G.: Single-trace fragment template attack on a 32-bit implementation of keccak. In: Grosso, V., Pöppelmann, T. (eds.) CARDIS'21. LNCS, vol. 13173, pp. 3–23. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-030-97348-3_1
26. You, S.C., Kuhn, M.G., Sarkar, S., Hao, F.: Low trace-count template attacks on 32-bit implementations of ASCON AEAD. IACR TCHES **2023**(4), 344–366 (2023). https://doi.org/10.46586/tches.v2023.i4.344-366