Garbled-Circuits from an SCA Perspective

Free XOR can be Quite Expensive...

Itamar Levi[1] and Carmit Hazay[1]

Bar-Ilan University, Ramat-Gan, Israel
(e-mail: {first.last}@biu.ac.il)

Abstract. Garbling schemes, invented in the 80's by Yao (FOCS'86), have been a versatile and fundamental tool in modern cryptography. A prominent application of garbled circuits is constant round secure two-party computation, led to a long line of study of this object, where one of the most influential optimizations is Free-XOR (Kolesnikov and Schneider ICALP'08), introducing a global offset Δ for all garbled wire values where XOR gates are computed directly without garbling them.
To date, garbling sachems were not studied per their side-channel attacks (SCA) security characteristics, even though SCA pose a significant security threat to cryptographic devices. In this research we demonstrate that adversaries utilizing advanced SCA tools such as horizontal attacks, mixed with advanced hypothesis building and standard (vertical) SCA tools, can jeopardize garbling implementations.
Our main observation is that garbling schemes utilizing a global secret Δ open a door to quite trivial side-channel attacks. We model our side-channel attacks on the garbler's device and discuss the asymmetric setting where various computations are not performed on the evaluator side. This enables dangerous leakage extraction on the garbler and renders our attack impossible on the evaluator's side.
Theoretically, we first demonstrate on a simulated environment, that such attacks are quite devastating. Concretely, our attack is capable of extracting Δ when the circuit embeds only 8 input non-linear gates with fifth/first-order attack Success-Rates of 0.65/0.7. With as little as 3 such gates, our attack reduces the first-order Guessing Entropy of Δ from 128 to $\sim$ 48-bits. We further demonstrate our attack via an implementation and measurements data over an STM 32-bit processor software implementing circuit garbling, and discuss their limitations and mitigation tactics on logical, protocol and implementation layers.

Keywords: Secure Computation · Garbled Circuits · Free-XOR · Side-channel analysis · Horizontal Attacks · Single Trace

1 Introduction

Background. Secure multi-party computation (MPC) allows a set of n parties P_i to jointly compute a function f on their inputs such that nothing beyond the output of that function is revealed. Privacy of the inputs and correctness of the outputs need to be guaranteed even if some subset of the parties is corrupted by an adversary. Such an adversary (either passive or active), is assumed to break into the devices of a subset of parties, learning their entire state which includes the their secret inputs to the computation and their internal randomness. Efficient and privacy-preserving multi-party computation is an important goal of modern society, organizations and computational platforms as a whole. Vast research effort was spent to provide protocols which are concretely efficient and practical; see [BCS19, HIMV19, YWZ20, BCO+21] for recent implementation efforts.

Garbled circuits, invented in the 80's by Yao [Yao86], have been a versatile and fundamental tool in modern cryptography with a large body of work building up on Yao's construction, providing optimizations, abstractions, variations, and indeed several new

applications of garbled circuits. One of the prominent applications of garbled circuits is constant round secure two-party computation protocols (2PC) [LP09]. Such a protocol proceeds by designating one party as a garbler and the other as the evaluator. The garbler generates an encoded version of the computed function (viewed as a Boolean circuit), using its private randomness. This is referred to as a *garbled circuit* (GC). The GC is then sent to the evaluator along with input labels corresponding to the garbler's input bits. Next, the garbler and evaluator engage in an oblivious transfer (OT) protocol, through which the evaluator receives the input labels corresponding to its own input to the function. Finally, the evaluator uses all the input labels and the GC to compute the function's output which it can share with the garbler. Garbled circuits have found many additional applications both in theory and in practice such as one-time programs [GKR08], KDM-security [BHHI10], verifiable computation [GGP10], multi-hop homomorphic computations [GHV10], zero-knowledge argument systems [JKO13] and more.

Secure garbling of circuits and corresponding ways of reducing the garbling size has been the aim of a long line of research, e.g., [BMR90, NPS99, KS08, LP09]. The most common paradigm for garbling a circuit operates at the gate level where for each gate in the circuit, each line in the truth table of the gate functionality is encrypted separately (this is also known as the 'gate-by-gate paradigm'). The garbling procedure follows by choosing a pair of labels for each wire in the circuit corresponding to the values 0 and 1. Then for each gate and each row of the truth table, two labels, one for each input wire, are used as the joint key for masking the output label. The underlying primitive for realizing this mask is a symmetric-key algorithm (*e.g.*, a pseudorandom function (PRF) or a correlation robust hash function) which yields extremely fast algorithms. This paradigm led to a long sequence of successful optimizations in computation and communication, that established garbled circuits as a practical tool for achieving 2PC; see e.g., [PSSW09, KMR14, ZRE15].

As it stands today, all garbling implementations have found expression in MPC environments, where both the garbler and the evaluator are part of a distributed computation that is typically carried out on powerful devices (such as cloud servers). However, with the technology outbreak, platforms such as advanced Systems-On-a Chip (SoCs), smartphones, Automotive, IoTs and standard desktops are powerful enough to execute efficiently such schemes. Specifically, with the growing popularity of applications that require dedicated hardware or use such platforms, which can benefit from garbling schemes and secure computation, designing and executing special purpose GC-hardware is inevitable [JKSS10, HJP13]. For some examples we can consider one-time programs and obfuscation [GKR08, JKSS10, KSNO17, ZCD+19], post-quantum signature schemes [JKO13, AHIV17, CDG+17, KKW18] and verifiable computation [GGP10, LMR22]. Nevertheless, such hardware devices are susceptible to side-channels attacks (SCA).

Note that it is always possible to evaluate leakage resiliency on the primitive level (such as permutation, encryption, hashing) with the assumption that if we protect these lower levels primitives, higher abstractions such as GCs, which are instantiated with such secure primitives, will be protected as well. Nevertheless, protection against side-channels attacks induces significant costs with respect to latency, energy etc. E.g., the masking countermeasure may introduce an overhead between $2x - 10x$ [GMK17, CGLS20, GM18] (as a function of the the security level required). Therefore adding such protections should be well justified. The folklore, which is further supported by our paper, is that the evaluator's algorithm is relatively protected against side-channel attacks [GKR08]. The garbler's algorithm, on the other hand, has not discussed to date as sensitive to side-channel attacks. In this paper we push forward on these aspects and show cases in which special care should be given to SCA and additional countermeasures should be considered.

To be concrete, side channel attacks enable distinguishing internal secret values manipulated by the hardware, exploiting the dependencies of secret-key dependent computations and some physically measurable quantities. This is natural as any computation requires

some form of energy, like switching (dynamic) currents of transistors and devices. This energy is then manifested in various noisy measurable quantities. In this work, we consider a setting where garbling schemes are embedded within a small chip that realizes the garbler's role. With our attacker model we assume it got hold of some side-channel access to the garber's device. We further assume that the adversary can eavesdrop public communication channels, such as the ones used for sending the circuit description, if not agreed upon, and public garbled values.

1.1 Our Contribution

One of the most influential optimization introduced for garbling schemes is the Free-XOR technique proposed by Kolesnikov and Schneider [KS08], which reduces the computation and communication resources. Namely, it allows XOR gates to be evaluated directly by the Boolean XOR of their input labels/keys. Consequentially, a considerable effort is put in order to decompose the circuit computation efficiently to a network with minimal non-linear operations (in GF(2)) on account of more linear ones [BCCM20, TSR+20, SZ13][1]. As it stands today, this optimization is embedded within all garbled circuits implementations. In their construction, Kolesnikov and Schneider proposed to choose only one label/key associated with each wire independently at random. Namely, if wire numbers i and j are the input wires to an XOR gate and k is the output wire number, then the keys for the input wires are set by the pair $\{Wp_{\cdot}^0, Wp_{\cdot}^0 \oplus \Delta\}$ where Δ is a global offset used across all wire labels in the circuit. Following this approach, the output label is the XOR result of the two input labels implying that it will be set either to $\{Wp_i^0 \oplus Wp_j^0\}$ or $\{Wp_i^0 \oplus Wp_j^0 \oplus \Delta\}$. For non-linear operations, e.g., ANDs as illustrate in Fig. 1b it is understood that Δ appears both within the input keys and plantexts (which correspond to the output keys). Security of Free-XOR follows due to a non-standard assumption of circular security [CKKZ12].

The Free-XOR optimization is our starting point, introducing the correlation we take advantage of in our side-channel attacks. More concretely, in this work we show that SCA attacks on the garbler's device are viable and devastating.

1.2 Technical Overview

In more details, the garbler generates the garbled circuit from the Boolean representation of a λ-gates circuit C, typically represented by AND and XOR gates. This is defined by the process where each wire w_i for $i < \lambda + m$ (where m is the input length), which can take up a Boolean value in $\{0, 1\}$, is labeled by two randomly generated n-bit values corresponding to the logical '0' and '1' values (where n typically serves as the security parameter). These are denoted by W_i^j for $j \in \{0, 1\}$. Assuming that all gates have fan-in 2, then their truth table includes 4 entries while enabling the recovery of one of the two possible garbled output labels without learning its semantic Boolean value. Naively, each gate is associated with 4 entries (or ciphertexts) [LP07] where each entry requires a symmetric-key operation denoted by $\mathcal{E}_{W_i^j, W_k^{j'}}(W_l^{j''})$ where $j'' = j \wedge j'$ for an AND gate. Followup optimizations (see Section 2.1), further reduce the number of sent ciphertexts.

Our technical contribution relates to building an attack which is based on the observation that garbling schemes utilizing a global secret Δ generate side-channel attacks sensitivity. Our attacks merge information from multiple time samples within a leakage trace and from multiple non-linear input-layer garbled gates, utilize public known labels and other publicly available information provided by the protocol. We demonstrate that such attacks are quite devastating; technically, capable of extracting Δ when the circuit embeds only 8 input non-linear gates with fifth/first-order attack Success-Rates of 0.65/0.7. With as little as 3 such gates, our attack reduces the first-order Guessing Entropy of Δ from 128 to

[1] interestingly, it is similar to the masked SCA context [BDMD+20]

$\sim$ 48-bits. We further demonstrate our attack via an implementation and measurements data over an STM 32-bit processor software implementing circuit garbling. We discuss their limitations in terms of miss-classification of leakages, needed in our attack procedure, and various other parameters of the attack, such as the enumeration complexity of the adversary. Lastly, we list a spectrum of possible countermeasures.

2 Technical Background

This section discusses garbled circuits and some recent advances. We further introduce necessary SCA attacks related information and highlight some advances we make use of in this work, namely Horizontal or single-trace attacks.

2.1 Garbling Schemes

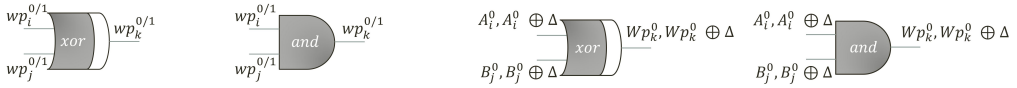
We recall the abstraction of a garbling scheme from [BHR12]. That is, a garbling scheme is a tuple of algorithms $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ where the probabilistic garbling algorithm Gb takes the function description f and outputs an encoded representation F and an input encoding function e. The deterministic input encoding algorithm En gets e and the function input x and returns an encoded input representation X. Finally, the deterministic evaluation algorithm Ev takes F and X and outputs $f(x)$ by evaluating the encoding. The classic Yao's garbled circuits construction follows by choosing two secret keys (also denoted as labels) for each circuit wire, each is associated with a single bit 0/1. The garbling is a process for which the Boolean truth table of each gate is "encrypted". In practice, this process is instantiated with symmetric-key cryptography e.g., pseudorandom functions (PRFs), double key PRFs or correlation robust hash functions.

The input to the circuit is obfuscated similarly. Finally, the evaluation process maintains the invariant that a single key per wire is obtained during the evaluation. This abstraction captures [LP09] and all its subsequent optimizations and is essential for proving security which follows by two properties of correctness and privacy, where the later ensures that the evaluator cannot learn anything but the output.

More concretely, to avoid revealing any information about the internal wire values, the table rows are typically permuted in a way which enables the recovery of the output labels without capturing their semantic values corresponding with '0'/'1'. The most practiced approach is the point-and-permute (PPR) technique [BMR90]. To achieve this, one more random bit is required for each wire label, i.e., each wire label is composed of $n+1$-bits, denoted here by: $Wp_i^j = \{W_i^j \| p_i^j\}, j \in \{0,1\}$, where $\|$ denotes concatenation and $p_i^j \in \{0,1\}$ is a random permutation bit. Applying the PPR optimization, the evaluator only needs to access one table entry per gate (rather than all 4 as illustrated in Fig. 1a). The garbled-row-reduction (GRR) [NPS99] is another optimization that enables to reduce the number of sent garbled values (ciphertexts) per gate from the 4 truth table entries outputs to three, improving on communication and computation costs.

Introducing the Free-XOR optimization, Kolesnikov and Schneider proved its security in the random oracle model [KS08] but also claimed that some form of correlation robustness would suffice. Following their work, Choi et al. [CKKZ12] have explored the security of garbled circuits with Free-XOR based on a stronger notion of circular correlation robustness and demonstrated that correlation robustness is insufficient as Free-XOR gives rise to related-key attacks which are not captured by correlation robustness.

With the aim of proving privacy under standard assumptions, Kolesnikov et al. proposed FleXOR gates [KMR14] (or flexible XOR) in which inputs can be adjusted to target specific Δ values. In cases input wires Δs' match with the output wire one, there is no need in such an adjustment, whereas in case only one of them matches, adjustment is required only on the corresponding input. The FleXOR optimization is also compatible with the advanced row-reduction technique by Pinkas et al. [PSSW09] (denoted here by GRR2)



Inputs	Garbled input	XOR output	Garbled XOR output	AND output	Garbled AND output
0,0	wp_i^0, wp_j^0	0	$\varepsilon_{wp_i^0,wp_j^0}(wp_k^0)$	0	$\varepsilon_{wp_i^0,wp_j^0}(wp_k^0)$
0,1	wp_i^0, wp_j^1	1	$\varepsilon_{wp_i^0,wp_j^1}(wp_k^1)$	0	$\varepsilon_{wp_i^0,wp_j^1}(wp_k^0)$
1,0	wp_i^1, wp_j^0	1	$\varepsilon_{wp_i^1,wp_j^0}(wp_k^1)$	0	$\varepsilon_{wp_i^1,wp_j^0}(wp_k^0)$
1,1	wp_i^1, wp_j^1	0	$\varepsilon_{wp_i^1,wp_j^1}(wp_k^0)$	1	$\varepsilon_{wp_i^1,wp_j^1}(wp_k^1)$

(a)

Inputs	Garbled input	XOR output	Garbled XOR output	AND output	Garbled AND output
0,0	A_i^0, B_j^0	0	$Wp_k^0 = A_i^0 \oplus B_j^0$	0	$\varepsilon_{A_i^0,B_j^0}(wp_k^0)$
0,1	$A_i^0, B_j^0 \oplus \Delta$	1	$Wp_k^0 \oplus \Delta$	0	$\varepsilon_{A_i^0,B_j^0 \oplus \Delta}(wp_k^0)$
1,0	$A_i^0 \oplus \Delta, B_j^0$	1	$Wp_k^0 \oplus \Delta$	0	$\varepsilon_{A_i^0 \oplus \Delta,B_j^0}(wp_k^0)$
1,1	$A_i^0 \oplus \Delta, B_j^0 \oplus \Delta$	0	Wp_k^0	1	$\varepsilon_{A_i^0 \oplus \Delta,B_j^0 \oplus \Delta}(Wp_k^0 \oplus \Delta)$

(b)

Figure 1: Garbled gates look-up table: (a) w/o Free XOR (b) w Free XOR optimization.

which utilizes polynomial interpolation techniques to reduce the number of communicated ciphertexts per garbled gate. Nevertheless, the GRR2 technique is not-compliant as is with the Free-XOR technique as the labels offsets are the result of interpolation and thus unpredictable. Choosing a specific tactic, i.e., Free-XOR or FleXOR based implementation is not always straightforward and highly depends on the circuit to be implemented.

Half-Gates were introduced by Zahur et al. [ZRE15], considerably improving communication cost of GCs. Their methodology eliminated the need of the polynomial interpolation of the GRR2 optimization while proposing means to evaluate AND gates with two ciphertexts (like GRR2) while being compliant with the Free-XOR optimization.

In what follow, we specifically discuss side-channels leakage in garbling schemes that share a global secret (i.e., Δ). *In that spirit, the alarming aspect is precisely that the most prominent GC optimization is what, in essence, makes GC SCA sensitive.*

2.2 Side Channel Attacks (SCA)

Side channel attacks have repeatedly demonstrated the sensitivity of implemented cryptographic schemes. As such, current National Institute of Standardization and Technology (NIST) competitions for future symmetric-key, e.g., Authenticated-Encryption [TMC+21] and public-key Post-Quantum schemes [AASA+20], are considering SCA security and its associated electronic cost factors as very important aspects. Successful single leakage-trace SCA attacks were shown possible for public-key encryption/ digital signatures schemes, both on hardware and software implementations [PPM17, PKH+21, KAA21]. Furthermore, lately remote/network SCA were demonstrated to be very threatening in: (1) embedded IoT devices, e.g., through radio channels from a far [GS15, SGMT18, CPM+18, CFS20], and (2) utilizing network-/device-reliability and monitoring infrastructure, such as temperature or voltage sensing [RPD+18, TBP20, MTH+21] e.g., on server cores.

In this section we briefly recall some of the necessary SCA basics and details needed for this manuscript on SCA security evaluation-metrics and horizontal/single-trace attacks.

In more details, side channel attacks enable the extraction of secret values manipulated by the hardware by exploiting the dependencies of secret-key dependent computations and some physically measurable quantity. Most reports focus on side-channel leakage measured through power or electromagnetic radiation channels, owing to their ease of access and rather high signal to noise ratio. These leakage samples originate mainly as physical outcomes of dynamic (switching) current dissipation of microelectronic devices. Correlation or Differential power analysis (C/DPA) [KJJ99, BCO04] are powerful side-channel attacks that follow a divide-and-conquer approach: an estimation on distinct parts of the key (denoted by sub-keys) takes place, called hypothesis, and these hypotheses are checked for correlations with the measured leakage from the device through multiple tests. Typically, various pre-processing techniques and tests are required to average out the noise and reveal underlying correlations between the hypothesized sub-key dependent internal

secret-value and the leakage measurement. Once a specific sub-key is extracted, the next sub-key can be extracted by the adversary utilising the same leakage measurements set, underpinning the strength of SCA attacks.

In simulated environments we typically estimate a leakage model for some intermediate values manipulated by the device. A leakage model is aimed at representing to some extent the actual physical behaviour measured in the SCA paradigm; i.e., the leakage owing to internal values manipulation. For a commonly practiced leakage model, which typically nicely represents software implementations leakage, we can consider the Hamming Weight (HW) leakage model by which an intermediate variable y of n-bits leaks: $\alpha \cdot \mathrm{HW}(y) + \beta + \mathcal{N}(\mu, \sigma)$. Where, the α factor may represent some scaling owing to (e.g.,) current dividers on the main power supply path originating from Maxwell and Kirchhoff's Current Law (KCL). The β factor may represent some DC offset owing to some noisy element in the electronic system or current consuming elements which are data-independent. These factors highly vary between different implementations. $\mathcal{N}(\mu, \sigma)$ is the modeled noise distribution owing to internal factors within the device and external parameters such as environmental influences. The noise can generally be of various distributions, may be additive or not and independently and identically distributed (IID) per time sample or not. Within simple first-order estimation models, we typically assume a Gaussian additive and IID noise.

The SCA adversary is modeled as one which can thus get access or perform such measurements. Clearly, it does not necessarily imply physical proximity to the device, as measurements can be broadcast through networks, extracted from near or far-field electromagnetic emanation channels, and can be completely passive [GS15, SGMT18, CPM+18, CFS20]. Similarly to traditional cryptographic models, we assume one end of the communication is exposed to an adversary: decryption-leakage measurements are associated with some known ciphertexts (or encryption-leakage samples with their associated plaintexts). SCA attacks, are typically not sensitive to the forward or backward direction of the attack, i.e., through plaintexts or ciphertexts. This excludes structurally asymmetric constructions such as authenticated encryption and tag-verification procedures.

Along the last few decades, the most powerful SCA attacks were statistical, meaning many such (plaintext / encryption-leakage) pairs were retrieved by the adversary prior to the attack procedure. Then, the adversary follows a modeling phase of a key-dependent internal computation in the algorithm, and further models the effect this value has on the encryption-leakage. Then, a statistical distinguisher is used, with the goal of eliminating the large measurements noise to extract the hypothetical key from the leakage samples (relaying on statistics and the laws of large numbers). In the following, we recall the necessary background on model-based and profiled attacks used in this paper.

2.2.1 Model-Based Attacks

In the next two subsections we do not yet relate to the garbled-circuits in the attack context. Specifically, let's assume a device that performs encryption and that any randomness used by the protocol is public (e.g., AES-CTR mode). Therefore, the following discussion starts with a simplified view without a garbler or an evaluator, i.e. a standard SCA attack. With conventional CPA [BCO04], we assume multiple measurements are available to the adversary under the same secret-key, e.g., considering a symmetric-key block-cipher instantiation. Then an attack works as follows:

Let $l_{x,k}$ be a leakage trace measurement under a plaintext/key $x = \{x_0||x_1||...x_{15}\}$/ $k = \{k_0||k_1||...k_{15}\}$ of n-bits where for example each x_i or k_i represents one byte of say n =128 bits. To perform a CPA attack, one should choose a target computation to hypothesize (in the case of an AES, typically the first round Sbox output). That is, some logical manipulation of the known plaintext (byte) by a deterministic function and the secret-key (byte) (in the case of an AES, the target intermediate value $y_i = \mathsf{Sbox}(x_i \oplus k_i)$). Once a specific intermediate value is chosen for an attack, the adversary builds an hypothesis table per each hypothesized secret-key $k_i \in \{0,1\}^8$ and all x_i related

possible measurements of size N_{tr} as abstractly illustrated in Fig. 2a. The leakage model for example can follow directly the HW of the intermediate hypothesized y_i value as discussed above. Next, a distinguisher can be used to find the hypothesized model which depends on the secret-key which most correlates with the measured leakage, e.g., a simple and commonly used distinguisher is the Pearson Correlation Coefficient:

$$\rho_{k^h} = \rho(\mathbf{y}_i^{k^h}, \mathbf{l}_{x,k}) = \frac{\text{cov}(\mathbf{y}_i^{k^h}, \mathbf{l}_{x,k})}{\sigma(\mathbf{y}_i^{k^h}) \cdot \sigma(\mathbf{l}_{x,k})} \tag{1}$$

where cov and σ are the covariance and standard deviation, respectively, and $\mathbf{y}_i^{k^h}$ is $\mathbf{y}_i$ computed under an hypothetical key k^h, i.e., $\mathbf{y}_i^{k^h} = \mathsf{Sbox}(x_i \oplus k^h)$.

Eventually, the secret-key (byte) k^* that maximizes the correlation is chosen, namely $k^* = \text{argmax}_{k^h}\ \rho_{k^h}$. In practice, each leakage trace is a vector over time $l^t_{x_i,k_i}$ where $t \in \{0, \ldots, \#\text{Samples}\}$ and the correlation in Eq. 1 takes per hypothesized key the maximum value found over time, estimated in a point-of-interest (POI) in time where the secret value y_i is being manipulated by the device [SMY09, LPB+15].

2.2.2 Template Attacks

An alternative approach for trying to model the leakage function of a device by apriori or a simple mathematical model, are template attacks [CRR02]. These are performed in two consequent (or interleaved) phases of *profiling* and *attack*. It is assumed that the adversary got hold of one device for which it can program (or control) the secret-key and therefore profile the leakage and another target device from which it tries to extract additional information on the underlying key.

Similarly to model-based attacks, to perform a template attack, one chooses a target intermediate variable to template the probability density function (PDF), $\mathsf{f}(x_i, k_i)$, associated with a known/chosen plaintext chunk x_i and the secret-key subset k_i at some POI. A set of $\mathcal{L}_p$ profiling traces of size N_p is first used in order to estimate the leakage distribution parameters for each intermediate value of y_i, denoted as $\hat{M}_{y_i}$. For the attack (online) phase, a fresh set of new traces from possibly a different device $\mathcal{L}_{att}$ of size N_{att} are used. In most cases in the literature (and in practice) the PDF is assumed to follow a Gaussian distribution, $\mathsf{f}(l_{y_i} \mid y_i) = \mathcal{N}\left(\widehat{\mu}_{l_{y_i}}, \widehat{\sigma}_{l_{y_i}}\right)$, i.e., in this case the statistical estimated model $\hat{M}_{y_i}$ contains a tuple $\{\widehat{\mu}_{l_{y_i}}, \widehat{\sigma}_{l_{y_i}}\}$. In practice, distributions can follow more complex structures, especially if countermeasures are embedded within the device. However, the simplicity of such a model, manifested by only two parameters per internal variable value, induces low model storage requirements, low computational effort and thus efficient, making it a very popular *de-facto* tool. Finally, the secret-key (byte/chunk) k^* is estimated to be the one which maximizes the uni-variate Maximum Likelihood (denoted by ML): $k_i^* = \underset{k_i}{\text{argmax}}\ \text{ML}(k_i) = \underset{k_i}{\text{argmax}} \prod_{i=1}^{N_{att}} (\mathsf{f}(l_i \mid y_i))$.

Due to practical computational reasons and numerical errors, the log-likelihood (LLH) is typically used [FDLZ14] $k_i^* = \underset{k_i}{\text{argmax}}\ \text{LLH}(k_i) = \underset{k_i}{\text{argmax}} \sum_{i=1}^{N_{\text{att}}} \log(\mathsf{f}(l_i \mid y_i))$.

It is noteworthy that many other (distinguisher,model)-pairs can be used instead of the ML distinguisher and the a-priori Gaussian modeled probabilities, *e.g.* a correlation distinguisher with a Hamming-Distance distinguisher, mutual-information based [GBTP08, PR10] or Moments-Correlating Profiled DPA (MCP-DPA) distinguishers [MS16][2].

2.2.3 Success Rate and Guessing Entropy

An evaluator typically uses a set of attack traces of size N_{tr} or N_{att} in the modeled or profiled settings, respectively, and computes a score vector per key based on the number of actual traces utilized out of the entire set. Based on the scores of the correct key and

[2] Which are very useful when masked implementations are considered with leakages manifested in higher statistical moments.

number of repetitions of the experiment, a Success-Rate and Guessing-Entropy can be computed. I.e., following an attack, the adversary obtains N_{sk} lists of sub-key rankings, where N_{sk} is the number of the sub-keys following the divide-and-conquer approach; 16 subkeys in the AES example above with an 8-bit length. In the case of profiled template attacks the adversary directly captures rankings in terms of probabilities (likelihoods). In the case of model-based correlation attacks, correlation values can be easily converted to probabilities, i.e., utilizing Base formulae or conversion formulae, for some examples see [SLP05, SMY09]. That is, each of the N_{sk} lists of size $|\mathcal{K}|$ (e.g., 2^8) contains a value, $\Pr[k_i|\mathbf{x}_i, \mathbf{l}_{x,k}]$. The rank R of the correct key is defined as its position in the ordered list of probabilities. The Success-Rate of order Ord, denoted by SR^{Ord}, being computed over N_{tr} leakage measurements per single experiment, is defined as [SMY09]:

$$\mathrm{SR}^{Ord}(N_{tr}) = \Pr[R < Ord]. \tag{2}$$

Simply put, SR^{Ord} is the probability that the correct key is ranked among the Ord first most probable keys. The Guessing Entropy (GE) can also be defined as the expected rank of the correct key directly linking the average enumeration effort of the adversary [SMY09].

2.3 Horizontal and Combined SCA Attacks

Horizontal attacks make use of hypotheses on variety of intermediate values within a single execution, i.e., multiple hypotheses of various intermediate variables which relate to the same secret value being hypothesized. Such attacks combine the collective informativeness/ correlations with multiple leakage samples to extract secrets.

Fig. 2a illustrates the classic CPA procedure where leakages from the same time-sample are jointly grouped (vertically) to a vector which is then correlated with the leakage-model

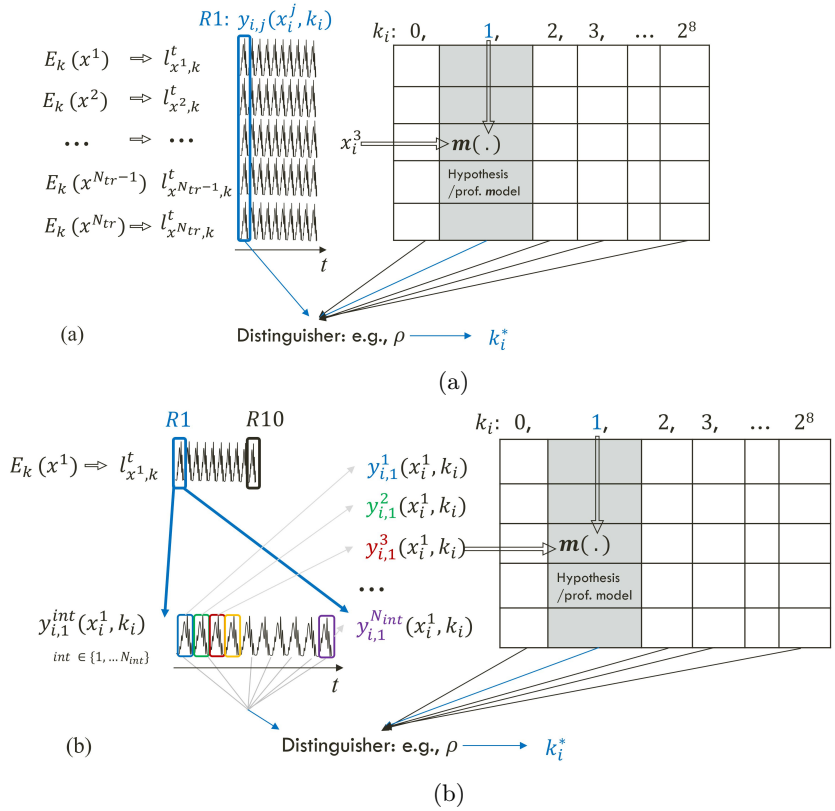


Figure 2: Conventional SCA apparatus: (a) CPA attack (b) Horizontal attack.

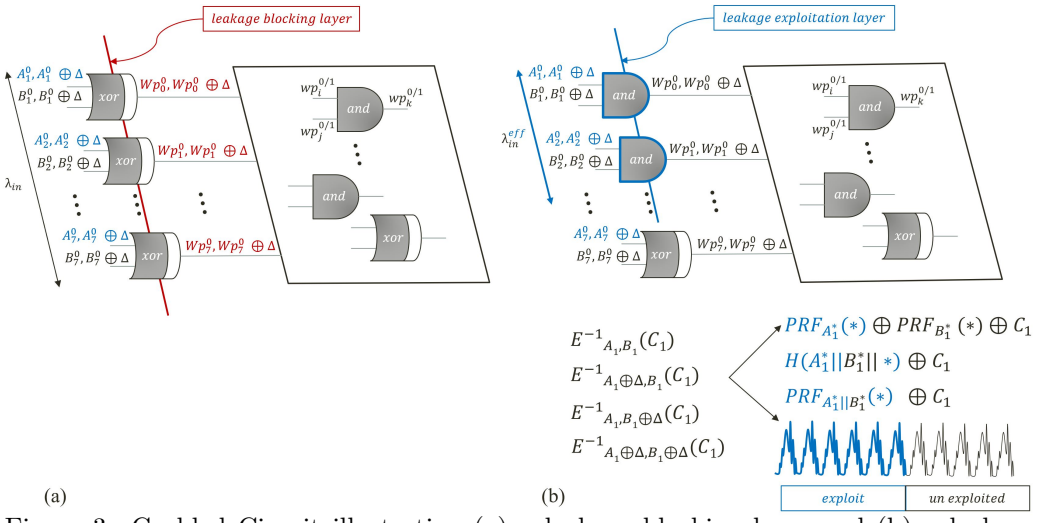
vector, denoted on the figure by $m(\cdot)$. This context is regarded here as *vertical* SCA (V-SCA). In this scenario, the adversary collects multiple measurements, each with possibly a different input indexed with a superscript x^i, and performs a single test on the power trace to extract sub-key correlations and filter out the noise as abstractly illustrated. A targeted intermediate variable y^i is one which its value depends on a public input (x^i) and an unknown sub-key.

In contrast, in *horizontal* SCA attacks, H-SCA [BJPW13, BCPZ16, PPM17, PZS17, KAA21, PKH+21], as illustrated in Fig. 2b, the adversary targets various computations under the same x^i and thus builds a model for multiple samples from the same leakage trace. Then these collective hypothesis are combined with the goal of extracting the sub-key from (e.g.) a single measurement as illustrated on the figure. Horizontal SCAs' focuses on multiple intermediate computations, $int \in \{1, \ldots, N_{int}\}$ that all depend on the public input and the same sub-key. The main challenge of applying horizontal SCA is in grouping multiple internal computations within a single computation, that leak over the same sub-key in a meaningful way (information sense), and to be able to nicely allocate the POIs each is associated with. Even though this attack is more complex, as an adversary needs to compute possibly more hypotheses over more internal variables and also needs to characterize their associated sets of POIs, it is more powerful and can theoretically extract more information or alternatively jeopardize the secret-key with far less measurements. In the context of asymmetric setting, randomized algorithms, re-keying methods and countermeasures, clearly such attacks are dangerous and thus important.

Combined Attacks. V-SCA and H-SCA can be combined. That is, multiple horizontal hypothesis can be utilized over multiple (vertical) traces collected while processing different data. In what follows below, our discussed attack merges V-SCA and H-SCA over the leakage extracted from gates garbling.

3 Attacking the Garbler and the Evaluator

In this section we detail on topological aspects which make the proposed attack viable. We assume that the adversary can monitor the communication channel and get a hold of the garbler associated labels (A_i^js'), and introduce the notion of leakage-blocking layer and leakage exploitation layer in garbled circuits as illustrated in Fig. 3. In this figure we consider a general garbled circuit which is comprised of multiple input-layer



(a) (b)

Figure 3: Garbled Circuit illustration (a) a leakage blocking layer and (b) a leakage exploitation layer.

gates and 'deeper' gates fanning-out from the input layer. Considering Fig. 3(a) which schematically illustrates some GC instantiation, one can identify the layer of input gates size λ_{in}. In this example all input gates are XOR gates; thus, considering a globally shared Δ. Nevertheless, as the XOR operation is locally computed not by cryptographic means and without communication, extracting some information from the leakage would be hard as the unknown B labels serve as a one-time-pad and the leakage is anyway very short in time (only one XOR operation) limiting the amount of information which can be extracted. Namely, the output labels are masked in the sense that an adversary cannot utilize them to further build a leakage model for subsequent gates. Thus, such a linear (XOR) input layer is considered here as a leakage blocking layer.

Considering Fig. 3(b) which schematically illustrates another GC instantiation, in this example a subset of the input gates are non-linear (e.g., ANDs) and are therefore being implemented based on a cryptographic object. Let the *effective* number of such gates in the input layer be denoted by λ_{in}^{eff}. As illustrated by prior work, the implementation of these non-linear gates by cryptographic means can often be partitioned into public and private parts where the former are known to the adversary (such as in computations that are based on the garbler's input labels set $\{A_i^j\}$). These can be processed and attacked separately (blue highlight), e.g., as in $\mathrm{PRF}_{A_1^*}(*) \oplus \mathrm{PRF}_{B_1^*}(*) \oplus C_1$ [NPS99] or $\mathrm{PRF}_{A_1^*||B_1^*}(*) \oplus C_1$ [KSS12], where A_1 and B_1 are the input labels, and C_1 is the output label. Namely, one part of the leakage will only depend on (blue) public labels and public data, and the rest of the leakage (black) depends on secret labels or a mixture. *These are the main openings which empower the discussed attack:* (1) incorporating a globally shared secret Δ as specified by Free-XOR and, (2) the fact that it is possible to partition the computation's leakage to several parts which are associated with public information only (public labels) and a shared secret (Δ), enabling building statistical model. *This is why such a non-linear (AND) input layer is considered here as a leakage exploitation layer.*

Before introducing our attack we recall that the adversary does not know the labels association and whether it knows A or $A \oplus \Delta$. This information is not needed in order to build the leakage profile as we build an hypothesis for it (see Section 2.2 for hypothesis definition in the SCA context); we refer to this specific point below.

3.1 Running Example, Observations and Further Notations

In this subsection we discuss our running example and further highlight some observations relating to the type of primitive used to implement the garbled gates output/truth-table. We consider Fig. 4 and refer to the discussion in the previous subsection on leakage exploitation layer. We assume that the adversary got hold of a (long) leakage trace taken from the garbling device, as illustrated on the top-left part of the figure where the public known values are highlighted in blue. Assuming that the circuit is known to the attacker, it can then follow with partitioning the (long) leakage to shorter snapshots, each associated with a non-linear input gate. Following this partition, the leakages are abstractly stacked one above the other[3], so as to be utilized in a conventional SCA attack, as illustrated in the lower-left part of the figure.

The literature proposes several different primitives to implement the cryptographic garbled-gate operation such as PRFs, double key PRFs and correlation robust hash functions. As different options are sensitive to the proposed attack, we do not wish to limit the discussion at this stage and therefore several such primitives are listed in the upper-right part of the figure. Our only restriction is that the underlying primitive follows the popular confusion and diffusion paradigm, where iterations of smaller permutations are used to construct the larger permutations. In this case, even if public and private information are bind by concatenation ($A_1^*||B_1^*$), it will still be trivially sensitive to SCA divide-and-conquer paradigm due to the parallel confusion layer.

[3] Or placed in an array, where each gate is associated with the leakage in a different row.

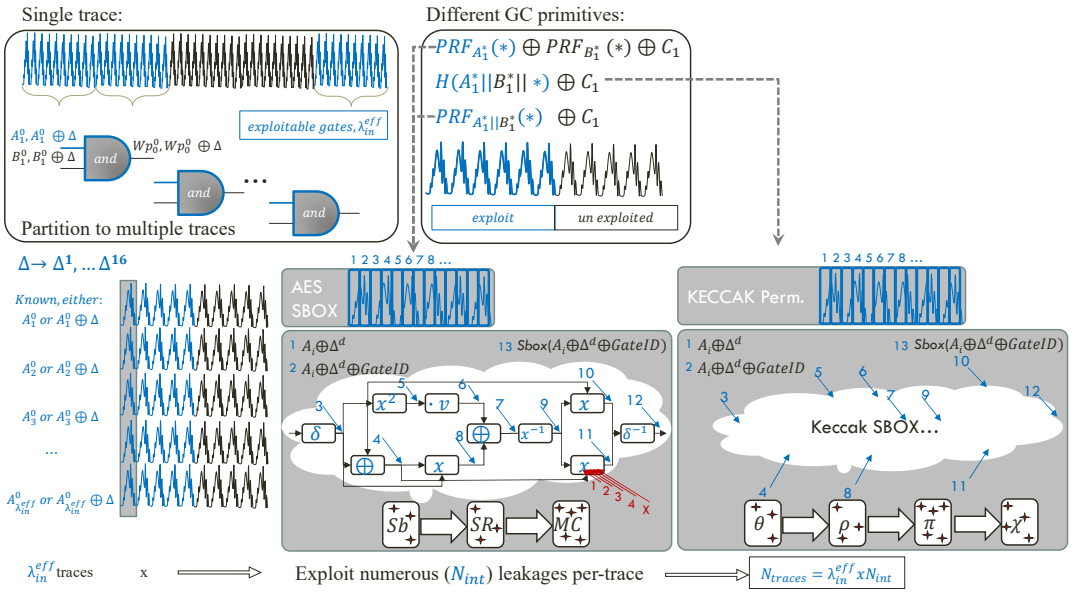


Figure 4: Attack illustration and internal leakages per trace, viable for advanced leakage model building (hypothesis). Where GateID denotes a public tweak known to the attacker.

Next we consider the popular and widely studied garbled-gate instantiation based on AES, which is a widely accepted symmetric encryption algorithm in modern IT, owing to its proven high security level, performance and support on virtually all platforms. We discuss an attack by a weak divide-and-conquer adversary targeting only an 8-bit enumeration complexity, namely growing with 2^8, and claim that even such a weak adversary is empowered by a large amount of information from the leakage traces. In our example, Fig. 4, and relating to the H-SCA discussion in subsection 2.3, an adversary can choose to exploit several internal computations within an AES. From the algorithmic perspective only two intermediate variables with 8-bit complexity exist in the first AES round before the code diffuses and is manipulated by other sub-keys, these are the XOR and the Sbox operations. Each of these variables depends on parts of the secret Δ and the garbler transmitted labels $\{A_i^j\}$. Our attack relies on the following fact: numerous leakage samples in various POIs can be extracted from correlations with these two internal variables alone. We next discuss the mechanisms which enable this property:

- **Algorithmic access -** many parts of the first-round computations are highly correlated with a selected Sbox output in various time-samples: a memory access to a lookup table it is stored in, its further manipulation in the shift-rows and mix columns stages of the algorithm being read from the register-file (RF), and small values manipulations such as shift which exist in mix-columns. These allow for multiple leakage samples with an independent noise.
- **Adversary's abilities -** when the sample-rate of the measurement equipment is high, many proximate points in time can leak significantly on the same variable.

In the H-SCA context, all the above mentioned leakages, associated with different time samples on the same internal variables, are profiled and used in the attack procedure. Below, in the experimental part of the paper, we show that by only targeting these two intermediate variables, each with associated sets of as little as 5 POIs (and up to 100), Δ can be easily recovered. Moreover, we point that other internal variables can be used to extract additional information. For example, taking the efficient and popular, tower-field $\mathbb{GF}(((2^2)^2)^2)$ AES-Sbox implementation, which does not utilize a memory look-up table for

the Sbox, but implements it by a sequence of Boolean operations: an adversary can make use of multiple internal computations which manipulate both the global secret Δ and the known garbler labels as illustrated in the bottom part of Fig. 4. In this case, it is easy to find at least 13 such 8-bit variables. It is worth noting that there are multiple other internal computations within the internal blocks of this Sbox such as the GF multiplier, inverter and squarer, which potentially leak. In that sense, each gate contributes multiple leakages which eventually aids the SCA adversary to average out the measurement noise, increase the SNR etc. Moreover, instead of targeting only 8-bits sub-keys in the divide-and-conquer approach, larger subkeys can be hypothesized leading to a larger set of internal variables independent in other subkeys, extracting more information per trace.

The above example is not only restricted to PRF-based constructions. The garbling instantiaion $\mathrm{H}_{A_1^*||B_1^*||*}(*) \oplus C_1$ [LPS08] that utilizes a hash function also suffers from a similar sensitivity. Considering the SHA-3 construction based on the KECCAK permutation, it is still trivial to identify the same scenario and leak numerous intermediate values in various POIs in the same fashion as illustrated on the bottom right part of the figure.

3.2 Attacking the Garbler

In the general case, the adversary captures the leakage from the garbler's hardware. This leakage is associated with a permuted garbled table $\mathcal{E}_{A_i^j,B_k^j}(W_l^j)$, $\mathcal{E}_{A_i^j\oplus\Delta,B_k^j}(W_l^j)$, $\mathcal{E}_{A_i^j,B_k^j\oplus\Delta}(W_l^j)$ and $\mathcal{E}_{A_i^j\oplus\Delta,B_k^j\oplus\Delta}(W_l^j)$, where the permutation is unknown to the adversary. Nevertheless, in the SCA context, only two specific leakages out of the four mentioned above are more natural to exploit in order to extract information from. In more details, the SCA hypothesis, namely the internal values computed while hypothesizing Δ^i, are computed as follows. The adversary receives either A_i^j or $A_i^j \oplus \Delta$ without knowing which of which and aims at associating this value with a leakage, viable to extract Δ. Therefore, in the case the received garbled value is A_i^j there would exist two options of correctly associating their leakage, namely, $\mathcal{E}_{A_i^j\oplus\Delta,B_k^j}(W_l^j)$ and $\mathcal{E}_{A_i^j\oplus\Delta,B_k^j\oplus\Delta}(W_l^j)$. Whereas, if the received garbled value is $A_i^j \oplus \Delta$, the two other garbled-table options would correctly associate leakages which are viable for information extraction. **The hypothesized value would anyway be the exclusive or of the received label with an hypothetical Δ.**

Therefore, an easy technique can be to exhaust all four options to correctly select these associated leakages. The success probability of doing that per gate is half and therefore the complexity for correctly associating the leakages of all λ_{in}^{eff} gates is $2^{-\lambda_{in}^{eff}}$ in total in the worst case[4]. The adversary utilizes each associated garbled label per non-linear gate to build a hypothesis. Generally, the attack is a conventional 'vertical' CPA augmented with multiple internal variables per-trace (horizontal attack components, of size N_{int}). Thus, the internal variables hypothesis (of the underlying cryptographic algorithm) can be kept of ultra low complexity depending on a sub-key of size $|\Delta^i|$. For example, the attacks in Section 5 below consider values of 8 and 16 bit-lengths. In this spirit, we denote our attack by a V/H-prefix for vertical/horizontal e.g., for a model-based CPA or Template based log-likelihood distinguisher we note V/H-CPA, V/H-TA.

The pseudo-algorithm of such an attack is listed in Alg. 1[5]. In the algorithm, first the long leakage trace associated with the entire computation the garbler performs is being measured. Then, this leakage trace is partitioned into smaller leakages associated with the particular gates in accordance with the topology of the circuit, where the adversary discards all but the leakages associated with the non-linear input gates. The global secret Δ is then partitioned into sub-keys (Δ^is'). In the next step, the adversary chooses one out of four leakages per each of the λ_{in}^{eff} gates, and runs the attack over each such selection and for all gates jointly. Assuming the adversary knows A_i^j, one of these choices must

[4]Where small λ_{in}^{eff} values are needed for a successful attack as demonstrated below, with λ_{in}^{eff}>3.
[5]The comparison Step 8 in Alg. 1 can be implemented by either LLH, correlation etc.

contain only leakages associated with computations of the form: $\mathcal{E}_{A_i^j \oplus \Delta, B_k^j}(W_l^j)$ for all j, this is the easiest attack scenario. Notably, the attack will also work if the leakage that is associated with the computation that is defined by $\mathcal{E}_{A_i^j \oplus \Delta, B_k^j \oplus, \Delta}(W_l^j)$ is chosen instead, owing to the leakage part associated with the sensitive joint computation highlighted in light blue as discussed above. An important aspect is that even if some percentage of the gates associated with the leakage samples are 'wrong', it does not imply that the attack fails. Therefore, the attack complexity in practice can be considerably reduced below the complexity described above. In the implementation and evaluation section below, we explore the success-rate (SR) of our attack as a function of the number of gates wrongly associated with leakages out of λ_{in}^{eff}; denoted by "Error in leakage selection" in percentage.

Algorithm 1 Attacking the Garbler

1: **Capture** a leakage trace from the garbler's device.
2: **Partition** leakage trace to garbling leakages associated with each gate.
3: **Pre-processing:** keep leakage traces associated only with non-linear input gates, there exist four such encryption-leakage traces per λ_{in}^{eff} gates.
4: **Pre-processing:** partition Δ to N_{sk} sub-keys $(\Delta^i, i \in \{1..N_{sk}\})$.
5: **Loop:** over Δ^i.
6: **Loop:** over a selection which comprises one leakage out of four leakages associated with each gate, $4^{\lambda_{in}^{eff}}$ such options.
7: **Model:** build a leakage model of all internal computations used to extract information by using all of the garbler labels sent via communication and gateID. In the model, hypothesize all possible Δ^i values of all sub-keys.
8: **Compare** the hypothesized leakage model vector (size $\lambda_{in}^{eff} \cdot N_{int}$) with the set of selected leakages.
9: **Store** comparison results per Δ^i and chosen leakages.
10: **Find:** per Δ^i search for best comparison amongst the possible chosen leakages.

3.3 The Conceptual Hardness of Attacking the Evaluator

Admittedly, by using similar tactics, attacking the evaluator via leakage is impossible. The reasons for that are:

1. First on a general note, less computation takes place on the evaluator's side, thus there exists less leakage and less information to extract and accumulate.
2. On the evaluator's side, not all the garbled gates associated computations from the gates truth-table are leaking. Generally, only one value, associated with one of the table entries, leaks. Moreover, differently from the garbler attack scenario, here the computation is directly performed on the "known" garbled values. Namely, the adversary does not have access to any leakage computed over the known labels masked with Δ for which building a hypothesis is possible.

4 Modeling and Simulated Attacks

This section deals with simulating, modeling and evaluating our attack's Success-Rate before we follow with the actual measurements and experimentation in Section 5. Our starting point is simulating the leakage. Our goal is to simulate a rather general setting, and for that purpose we set some assumptions:

- As discussed in Section 3 and illustrated in Fig. 3b we do not restrict the structure of the garbled circuit and we set the leakage exploitation layer size λ_{in}^{eff} to 8. This parameter setting implies that only $8 \cdot 4$ AES leakage traces are captured by the adversary, thus highlighting the strength of our attack. In the experimental section,

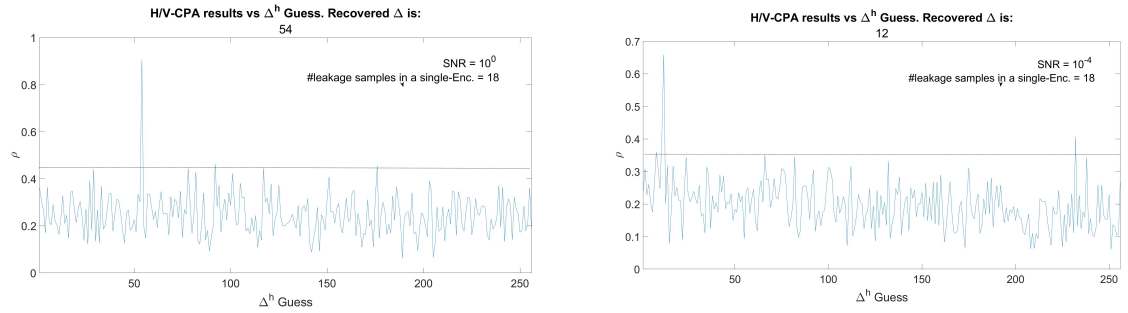
we show that this value can be much smaller for the attack to succeed in practice.
- We assume the garbled gate scheme utilizes PRF which is instantiated by AES. In particular, we follow the garbling construction $\mathrm{PRF}_{A_1^*}(*) \oplus \mathrm{PRF}_{B_1^*}(*) \oplus C_1$ [NPS99]. Nevertheless, as discussed above we believe that the same results and conclusions hold for other the constructions based $\mathrm{PRF}_{A_1^*||B_1^*}(*) \oplus C_1$ [KSS12], or $\mathrm{H}_{A_1^*||B_1^*||*}(*) \oplus C_1$ [LPS08], where (*) denotes a public knowledge.
- We assume that the garbler's device is under an attack and that the leakage samples are captured by monitoring it as well as exploiting the communicated garbler labels.

In simulating the leakages, we have used standard AES software packages to generate AES internal computation and exported internal states and various internal variables computed within the algorithm. The 8 AES-based garbled gates were asserted by randomly chosen labels and one joint Δ. As conventional with the side-channel divide-and-conquer approach, we partition Δ into N_{sk} sub-keys ($\Delta^i, i \in \{1..N_{sk}\}$). Per computed intermediate variable, we have computed its Hamming Weight (HW) as the noiseless leakage model and added a Gaussian noise with an SNR parameter [Man04] to represent the leakage as discussed in Section 2.2. The number of internal-variables used in the attack per each of the above mentioned 8 gates is denoted by N_{int} as defined above. We have experimented with various scenarios and different N_{int} values, some examples are listed below:

1. With N_{sk}=16 we build a hypothesis over 8-bit secret Δ chunks. Therefore, the natural approach would be to take N_{int}=2 modeling the first AES round XOR and Sbox output. Nevertheless, as discussed in Section 3.1 and shown in Section 5, in practice there exist numerous leakage points associated with either of these internal values, and each internal variable correlates greatly with various time samples in a leakage trace. For example, in case of a memory-based look-up table implementation of an Sbox, many time samples can all be used to extract information and effectively filter out the noise, handling values in RF and utilizing them in the shift-rows or mix-columns stages. All these steps, occur in different clock cycles many times.
2. With N_{sk}=8 we build a hypothesis over 16-bit secret Δ chunks. Therefore, we can build various hypotheses of several 8-bit XORs', Sbox outputs and various internal processing in the shift-rows and mix-columns stages in the first round which combine these values. Practically, allowing N_{int} value between 1 to 10 as listed in the experimental section below.

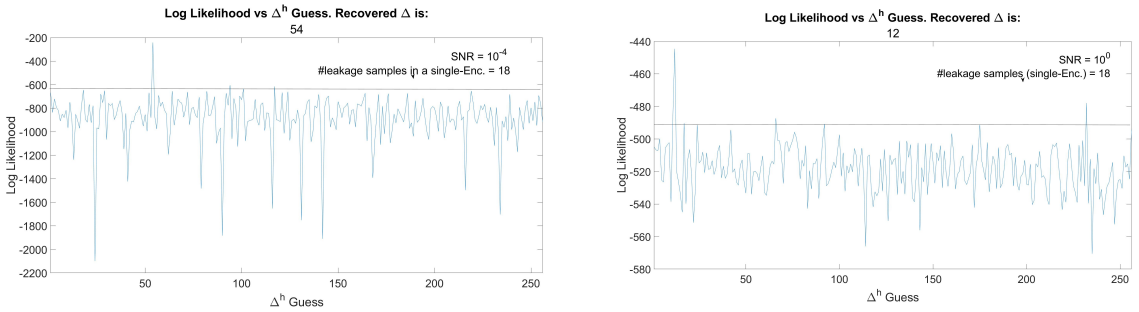
We next discuss the results from a synthetic example which remarkably agree with the experimental section below. Our goal here is to provide a general discussion of our simulation. In our example, for simplicity, we have assumed scenario (1) from the list above. This scenario ideally implies N_{int}=2. However, to comply with real-life scenarios, where leakage samples from multiple time instances correlate with internal variables, we have duplicated each of these two internal variables (per trace) several times, and added a fresh noise per each of these duplicated variables. E.g., instead of modeling some leakage of an internal value y by $HW(y)+n$, we have repeated the leakage from the same internal value with fresh randomness, $HW(y)+n_1, \ldots, HW(y)+n_r$. In the figures below, N_{int} or verbally, the *#leakage samples in a single encryption trace*, encompass these samples as well.

We have utilized the following two discussed distinguishers over the simulated traces:
- Model-based H/V-CPA attack with HW leakage model as discussed in Section 2.2. This attack is augmented with modeling Horizontal leakages with various N_{int} values, as explained in Section 2.3. Our λ_{in}^{eff} parameter implies that the number of attack traces used in our setting is N_{att}=8 and no profiling traces were used.
- Profiled Gaussian Template attacks with a modeled mean and standard-deviation as discussed in Section 2.2. This attack is augmented with profiled horizontal leakages as explained in Section 2.3. In this case, the number of attack traces is also N_{att}=8. However, relating to the profiled attack apparatus discussed in Section 2.2.2, we now



(a) (b)

Figure 5: Simulated attacks: (a) H/V-CPA with SNR=1 (b) H/V-CPA with SNR=10^{-2}.



(a) (b)

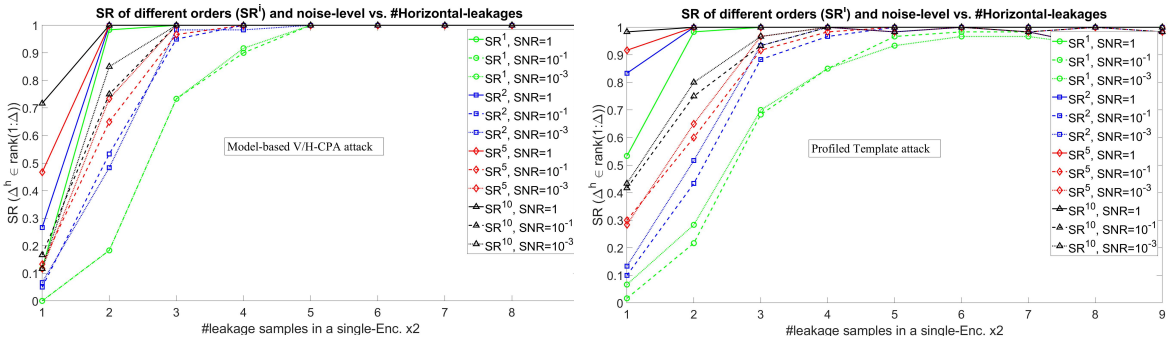
Figure 6: Simulated attacks: (a) Log-likelihood Profiled Template attack with SNR=1 (b) Log-likelihood Profiled Template attack with SNR=10^{-2}.

assume that the adversary can profile a similar device to model the leakage samples and find POIs. The number of profiling traces was in the range 100-1000, all showing successful results. Here we show results with N_p=500.

Fig 5 shows the advanced H/V-CPA attack results of a single 8-bit Δ^i. Fig 5a shows the correlation peak with the hypothetical correct Δ^i with high SNR level of 1, approaching ρ=1, whereas Fig 5b shows still the correlation peak with the hypothetical correct Δ^i with a low SNR level of 10^{-2}, reducing ρ's peak of the correct value. Fig 6 demonstrates the advanced log-likelihood based distinguisher Gaussian template attack results of a single 8-bit Δ^i. That is augmented with many horizontal modeled leakages. Fig 6a shows the correct Δ^i peak with high SNR level of 1 and Fig 6b shows still the correct Δ^i peak with a low SNR level of 10^{-2}. In both examples, N_{int} was chosen to be 18. Nevertheless, as shown below, in software scenarios this value can be far higher[6], making the attack much easier. In particular, the interesting question we were facing was "what is the bare-minimum number of leaking time-points per internal variable modeled, required for a successful attack?" That is, how many internal *horizontal* leakage samples suffice?

To answer this question we have repeated our modeled-attacks with a varying N_{int} and a varying SNR level. Each experiment was repeated 200 times with random garbler labels and a random Δ, where the attack Success-Rate of order Ord (SR^{Ord}) was computed for different orders. Fig. 7a shows the success rates of the V/H-CPA attack. It is possible to capture from the figure that first, with as little as 10 effective leakage samples per trace all attacks succeed, i.e., 5 leakage samples were correlated with each of the 8-bit XOR and Sbox for each of the 8 garbled gates. That is, totaling to 80 leakage samples, all attacks

[6]We evaluated the SR of the same attack with up to 100 internal leakages per internal variable in a measured trace.



(a) (b)

Figure 7: Success Rate of order o (SR^o) with varying Horizontal information per encryption ($N_{int} \in \{1, \ldots, 18\}$): (a) Model-based V/H-CPA (b) Profiled Gaussian Template.

are successful with probability of 1. With only two samples, i.e., one of the XOR and one of the Sbox, the attack effectively reduces the entropy of the Δ^i to 3 bits out of 8. Clearly progressively smaller orders, SR^{Ord}, converge with more samples.

Fig. 7b shows our results from the same experiment over the profiled template attacks, showing similar trends. Nevertheless, it is possible to see that with a smaller number of samples (horizontal leakage points in the x axis), the attack achieves higher success rates owing to the better (profiled) leakage model. Clearly, we can expect that in the experimental part of the paper the profiled setting will be much more powerful as compared to the model-based CPA attack, as actual leakage functions of devices do not completely follow the HW leakage model.

5 Implementation and Attacks

In this section we follow with concrete measurements data and remove synthetic modeling assumptions from our attacks. We utilize measurements from an STM 32-bit software implementation of Tiny-AES while garbling the circuit. We follow with the same example of an exploitable leakage layer of eight non-linear gates. Our measurements-related assumptions are as follows:

- **Synchronization** - Following a circuit garbling we assume the adversary can perfectly "cut" independent leakages in a synchronized fashion. That is, we eliminate jitter and provide a clean and aligned traces scenario. The motivation to evaluate this scenario corresponds with the high-SNR typically perceived in software leakages. This makes (unprotected) leakages rather easy to decently align, i.e., with techniques like elastic alignment [vWWB11] and cross-correlation [LCWY09]. We further discuss below cases of clock jitter/drifts, software-interrupts or deliberate shuffling countermeasures.
- **Permutations** - In the first part below we assume the adversarial best case, namely, it specifically associates the correct exploitable leakages out of each possible 4-tuple leakages per gate. Later, we further evaluate error-rates regarding this assumption.
- **Sniffed labels** - We assume the adversary got hold of the needed associated labels.

Random labels and random Δs' (but fixed per attack instantiation) were generated in our attack using measurements from a Tiny-AES software implementation over the New-AE Chipwesperer Pro with an STM32F target device taken from [B+19]. Before discussing the results, we first illustrate the mean leakage of 200 traces as shown in Fig. 8. Clearly, AES rounds are visible within the leakage.

In this experimental part of the paper we have evaluated two attack scenarios (1) N_{sk}=16 with an 8-bit attack and (2) N_{sk}=8 with a 16-bit attack. We start with discussing the first case. We have performed both modeled and profiled attacks. For both the profiled

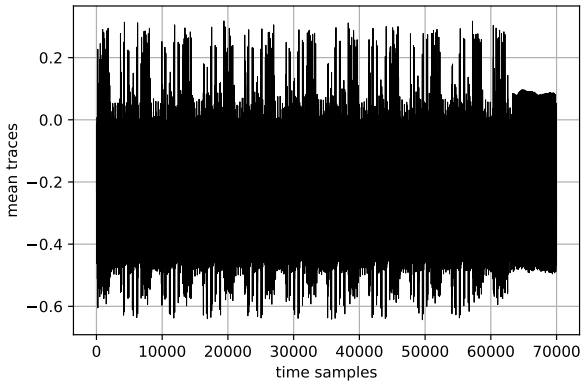
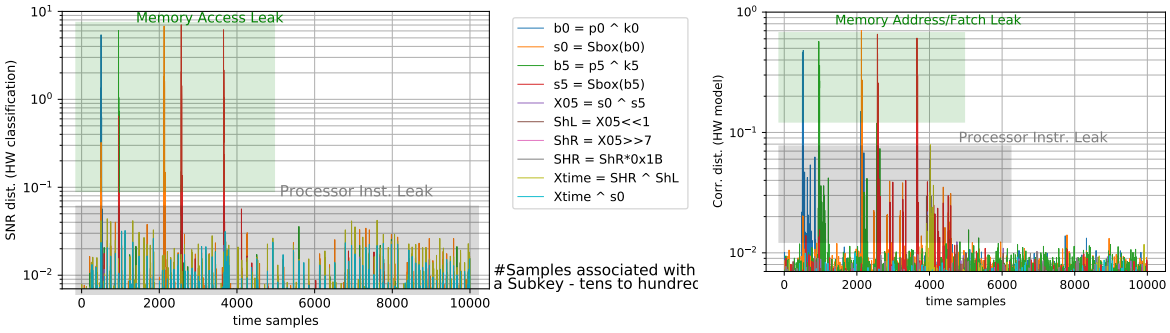


Figure 8: Mean leakage of a Tiny-AES, software STM 32-bit processor.



(a) (b)

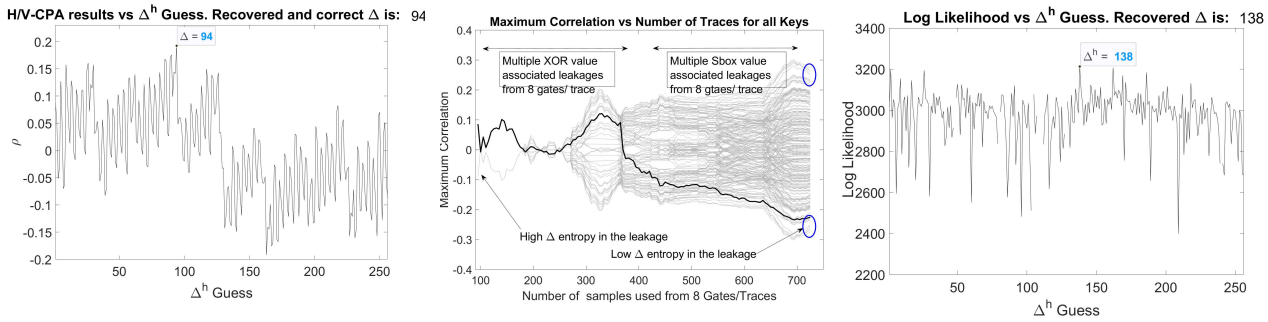
Figure 9: Zoomed-in 1^{st} AES round of a Tiny-AES software running over an STM 32-bit processor. Leakages feature numerous samples associated with internal variables depending on a single key-byte: (a) SNR with HW classification (b) Correlation distinguisher.

and non-profiled attacks the adversary needs to find sets of POIs per internal variable. To get these, we used up to 1000 traces with random input data and compute both the SNR [Man04] and the univariate correlation [BCO04], to be used as distinguishers to identify these POIs' sets per internal variable. Fig. 9a and 9b shows the resulting SNR and correlation, respectively over a zoom-in to the first AES round leakage. In this illustration, for exposition purposes, we show SNR and correlation trends for various internal-variables as listed below for the more comprehensive 16-bit attack case.

Modeled (hypothesized) internal variables in the 16-bit $|\Delta^i|$ case are:

1. $x_0 = A_0^* \oplus \Delta^0 \oplus (*)$ equivalent in the AES to the computation: $p_0 \oplus k_0$.
2. $s_0 = \text{Sbox}(A_0^* \oplus \Delta^0 \oplus (*))$ equivalent in the AES to the computation: $\text{Sbox}(p_0 \oplus k_0)$.
3. $x_4 = A_4^* \oplus \Delta^4 \oplus (*)$ equivalent in the AES to the computation: $p_4 \oplus k_4$.
4. $s_4 = \text{Sbox}(A_4^* \oplus \Delta^4 \oplus (*))$ equivalent in the AES to the computation: $\text{Sbox}(s_4 \oplus s_4)$.
5. $xs_{0\_4} = (s_4 \oplus s_0)$ equivalent AES computation.
6. $SL = 2 \cdot xs_{0\_4} \bmod 2^8$ representing the modulus shift left in the Tiny AES.
7. $SR = \lfloor xs_{0\_4}/2^8 \rfloor$ representing the shift right in the Tiny AES.
8. $Val1 = SR \cdot 27$ representing the conditional primitive polynomial reduction.
9. $Val2 = SL \oplus Val1$ representing the mix columns internal computation.
10. $Val3 = Val2 \oplus s_4$ representing the mix columns internal computation.

Where, as discussed above (*) represents public information, A^* represents one of the gates' labels and A_0^* indicates the zero byte chunk of this label, and p_0 represents the



(a) (b) (c)

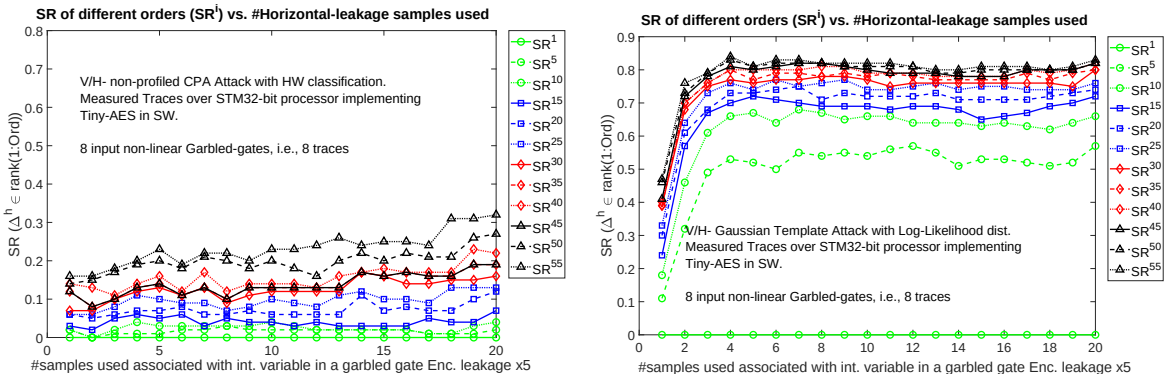
Figure 10: Attacks results over a Tiny-AES SW implementation running on an STM 32-bit processor, examples: (a) H/V-CPA vs. Δ guess (b) H/V-CPA vs. the number of samples used by the attack (c) Log-likelihood Profiled Template attack vs. Δ guess.

plaintext chunk. Per each of these internal values 100 POIs' were evaluated, this step can be easily profiled with as little as a few hundreds of traces with the SNR or correlation distinguishers and with other simple and univariate dimensionallity reduction tools.

From Fig. 9 it is easily captured that numerous leakage samples correlate with the modeled internal-values pinpointing and highlighting the power of horizontal-attacks. For example, in the Tiny-AES implementation, as AES Sbox is stored in memory, leakages associated with memory access are clearly much leakier showing far higher SNR/Correlation levels. However, numerous other leakage samples associated with the same internal-variables leak in a round associated with Register-File access and ALU computations on these values (for example mix-columns operations). Though we have experimented with up to 100 samples per internal variable, a higher number of samples may exist. Furthermore, if one further zooms-in to each of these peaks in the plot, it is possible to see that it is actually smeared over time (depending on the sample-rate of the oscilloscope and the adversaries capabilities). This means that every single operation/computation leaks in many proximate samples which can enable repeated sampling and averaging. The sets of POIs ware stored and utilized in the attack procedure.

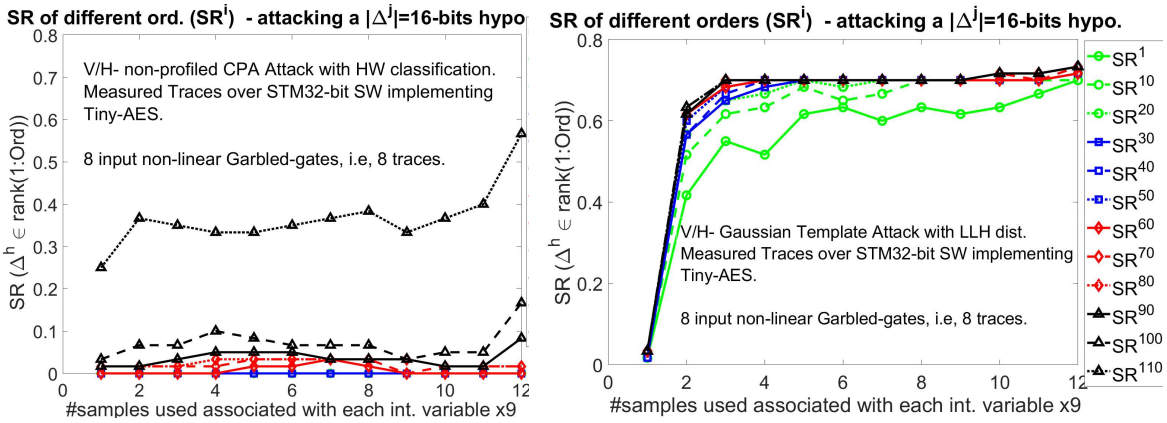
Fig. 11 shows results from a single attack. That is, Figures 11a and 11c show the results from an 8-bit H/V-CPA and log-likelihood template attack, respectively. As shown Δ^is' are correctly identified. In these cases we took N_{int}=100 just for illustration (i.e. maximum). The success-rates while varying N_{int} are discussed below. Fig. 11b shows an interesting illustration of the H/V-CPA attack while varying N_{int}. That is, the figure shows the correlations of all hypothetical Δ^is' and from the left with no horizontal leakages, to the right with maximum number of leakage samples: # horizontal leakages (80 in this case) $\cdot$ # of Gates (8) $\cdot$ # of internal variables (2, XOR and Sbox). The figure highlights the reduction in guessing entropy of the key as more and more information is extracted from the leakage in our H/V apparatus. We have followed with computing the success rates of these attacks of both the modeled and profiled attacks and for both the 8-bit and 16-bit attack scenarios. Figures 11 and 12 show the 8-bit and 16-bit scenarios, respectively. Each of these figures shows the model-based V/H-CPA case to the left (i.e. (a)) and the log-likelihood profiled Gaussian-Template attack case to the right (i.e. (b)). Quite different from the results of the modeling section, here we capture a rather poor behaviour of the model-less attacks which only provides significant Δ^i entropy reduction with maximum number of horizontal leakages (N_{int}) or alternatively while hypothesizing 16-bits internal variables which enable hypothesis building of more internal variables in the algorithm.

The profiled Gaussian-Template attack shows a remarkable difference illustrating very high success rates with as little as 4-6 horizontal leakages per internal variable even for



(a) (b)

Figure 11: Attack Success Rate of various orders Ord (SR^{Ord}) with varying Horizontal information per encryption ($N_{int} \in \{1, \ldots, 100\}$), Profiled Gaussian Template based attacks. Attack over $|\Delta^j| = 8$ bits: (a) V/H-CPA (b) Log-likelihood Gaussian Template.



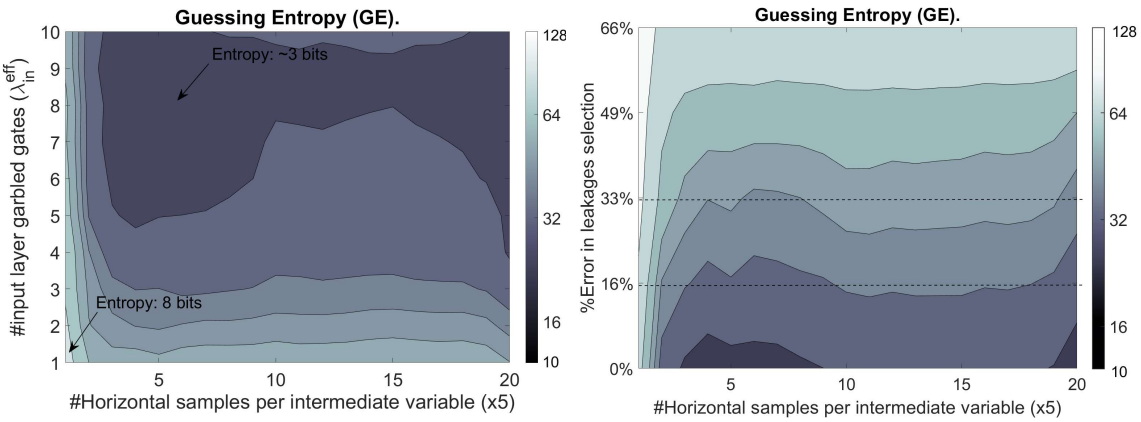
(a) (b)

Figure 12: Attack Success Rate of various orders Ord (SR^{Ord}) with varying Horizontal information per encryption ($N_{int} \in \{1, \ldots, 100\}$), Profiled Gaussian Template based attacks. Attack over $|\Delta^j| = 16$ bits: (a) V/H-CPA (b) Log-likelihood Gaussian Template.

very low orders indicating the efficiency and viability of the garbler attack.

An interesting question we were facing was "what is the size of the minimal leakage exploitation layer?" Namely, what is the minimum number of non-linear input gates λ_{in}^{eff}, enabling an attack. We have implemented such an attack varying λ_{in}^{eff} from 1 to 10 gates. Fig. 13b shows the guessing-entropy (GE) of Δ^i versus λ_{in}^{eff} (y axis) and the number of horizontal leakage samples utilized in the attack. The GE value is contoured with a color-map showing that with as little as 5 horizontal samples per intermediate variable and even with three input gates, concrete reduction in entropy of Δ^i is achieved with 4-bits, as compared to the lower-left corner (i.e., full 8-bit entropy). Therefore, even a very low number of non-linear input garbled gates can be quite dangerous.

Finally, in this experimental part of the manuscript we were interested to evaluate the success-rate (or guessing-entropy) reduction owing to the fact that the adversary does not know whether the garbler's associated labels, captured via the communication interface, reflect A or $A \oplus \Delta$. That is, we were interested to understand the influence of 'wrong' gate's associated leakages. The specific question we were interested to answer was "weather



(a) (b)

Figure 13: Order-1 Guessing Entropy (GE^1): (a) vs. the number of the non-linear garbled gates (i.e., the size of the leakage sensitive input layer, λ_{in}^{eff}) (b) vs. the Error in correctly selecting $A_j^i \oplus \Delta$ associated leakages out of λ_{in}^{eff}.

it implies an unsuccessful attack and at what stage?" If it does not considerably lower the attack success-rate, the attack complexity in practice can clearly be considerably lower than the complexity described in Section 3.2. In Fig. 13b we explore the guessing-entropy as a function of the number of wrongly associated gates out of λ_{in}^{eff}; denoted by "Error in leakage selection", in %. It is possible to see that even with 16%/32% erroneous gates, significant entropy reduction is still achieved, i.e. 4-/5-bits, respectively.

6 Future Research and Countermeasures

In this paper we have substantiated that if one aims to consider the attractive Free-XOR optimization for garbled circuits, SCAs are trivially viable and countermeasures are required. Below, we list potential future-research directions and various countermeasures, at the construction and protocol levels and the hardware/software implementation level:

- **Binding:** bind both garbler's and evaluator's garbled keys. One such approach was demonstrated by Bellare et al. in [BHKR13] by proposing the garbled gate operation as $\mathcal{E}_{A_i^j,B_k^j}(W_l^j) = PRF(const, 2A_i^j \oplus 4B_k^j \oplus gateID) \oplus 2A_i^j \oplus 4B_k^j \oplus gateID \oplus W_l^j$.
- **Protocol:** adding a leakage blocking layer as discussed in Section 3 by (e.g.) introducing XOR gates at all input ports leading to non-linear gates. Either by manipulating the logic or by tying one of the inputs of the inserted XOR gates to a logical garbled '0'. This tied input should be associated with the evaluator's input as well, thus preventing the adversary from propagating known inputs labels.
- **Re-keying:** amongst different gates, some form of a "re-keying" mechanism may be embedded in the sense that it should be hard to combine information from leakage samples associated with multiple gates. This could be achieved by: (1) rekeying using a Nonce-based mode (2) adding some random tweak-bits. Such solutions would clearly come at a cost, as this randomness should be obliviously communicated.
- **Implementation countermeasures:** clearly, embedding Hardware/Software Masking [ISW03, DCEM18, GMK18, BDMD+20, CGLS20] may make the already huge relative cost of GCs far larger. However, low-order masking approaches with or without other simple countermeasures such as instruction-shuffling [VCMKS12, LBS20] or amplitude-randomization techniques [LBBS20] would render attacks impractical, due to the relatively low number of observations captured by the adversary.
- **Protocol level shuffling:** another approach we identify as interesting would be to

randomly shuffle the gabled gates execution at the protocol level while keeping the shuffling key a secret. The goal would be to increase the attack complexity and the error-rate for incorrectly associating leakage, potentially exponentially with $|C| = \lambda$.

References

[AASA+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the second round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2020.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104, 2017.

[B+19] Elie Bursztein et al. Scaaml: Side channel attacks assisted with machine learning, 2019.

[BCCM20] Anna Bernasconi, Stelvio Cimato, Valentina Ciriani, and Maria Chiara Molteni. Multiplicative complexity of autosymmetric functions: Theory and applications to security. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

[BCO+21] Aner Ben-Efraim, Kelong Cong, Eran Omri, Emmanuela Orsini, Nigel P. Smart, and Eduardo Soria-Vazquez. Large scale, actively secure computation from LPN and free-xor garbled circuits. In *EUROCRYPT*, pages 33–63, 2021.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the isw masking scheme. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 23–39. Springer, 2016.

[BCS19] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using topgear in overdrive: A more efficient zkpok for SPDZ. In *SAC*, pages 274–302, 2019.

[BDMD+20] Begül Bilgin, Lauren De Meyer, Sébastien Duval, Itamar Levi, and François-Xavier Standaert. Low and depth and efficient inverses: a guide on s-boxes for low-latency masking. *IACR Transactions on Symmetric Cryptology*, 2020(1):144–184, 2020.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In *EUROCRYPT*, pages 423–444, 2010.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE, 2013.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, pages 784–796, 2012.

[BJPW13] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and vertical side-channel attacks against secure rsa implementations. In *Cryptographers' Track at the RSA Conference*, pages 1–17. Springer, 2013.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *CCS*, pages 1825–1842, 2017.

[CFS20] Giovanni Camurati, Aurélien Francillon, and François-Xavier Standaert. Understanding screaming channels: From a detailed analysis to improved attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 358–401, 2020.

[CGLS20] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 2020.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-xor" technique. In *TCC*, pages 39–53, 2012.

[CPM+18] Giovanni Camurati, Sebastian Poeplau, Marius Muench, Tom Hayes, and Aurélien Francillon. Screaming channels: When electromagnetic side channels meet radio transceivers. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 163–177, 2018.

[CRR02] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[DCEM18] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware masking, revisited. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 123–148, 2018.

[FDLZ14] Yunsi Fei, A Adam Ding, Jian Lao, and Liwei Zhang. A statistics-based fundamental model for side-channel attack analysis. *Cryptology ePrint Archive*, 2014.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 426–442. Springer, 2008.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. *i*-hop homomorphic encryption and rerandomizable yao circuits. In *CRYPTO*, pages 155–172, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[GM18] Hannes Gross and Stefan Mangard. A unified masking approach. *Journal of cryptographic engineering*, 8(2):109–124, 2018.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, pages 95–112, 2017.

[GMK18] Hannes Groß, Stefan Mangard, and Thomas Korak. *Domain-Oriented Masking–*. PhD thesis, Graz University of Technology, Austria, 2018.

[GS15] Gabriel Goller and Georg Sigl. Side channel attacks on smartphones and embedded devices using standard radio equipment. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 255–270. Springer, 2015.

[HIMV19] Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. Leviosa: Lightweight secure arithmetic computation. In *CCS*, pages 327–344, 2019.

[HJP13] Simon Hoerder, Kimmo Järvinen, and Daniel Page. On secure embedded token design. In *IFIP*, pages 112–128, 2013.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Annual International Cryptology Conference*, pages 463–481. Springer, 2003.

[JKO13] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *CCS*, pages 955–966, 2013.

[JKSS10] Kimmo Järvinen, Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs - (full version). In *CHES*, pages 383–397, 2010.

[KAA21] Emre Karabulut, Erdem Alkim, and Aydin Aysu. Single-trace side-channel attacks on ω-small polynomial sampling: With applications to ntru, ntru prime, and crystals-dilithium. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 35–45. IEEE, 2021.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *CCS*, pages 525–537, 2018.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. Flexor: Flexible garbling for XOR gates that beats free-xor. In *CRYPTO*, pages 440–457, 2014.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, pages 486–498, 2008.

[KSNO17] Takuya Kitamura, Kazumasa Shinagawa, Takashi Nishide, and Eiji Okamoto. One-time programs with cloud storage and its application to electronic money. In *ASIACCS*, pages 25–30, 2017.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *USENIX*, pages 285–300, 2012.

[LBBS20] Itamar Levi, Davide Bellizia, David Bol, and François-Xavier Standaert. Ask less, get more: Side-channel signal hiding, revisited. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2020.

[LBS20] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. Beyond algorithmic noise or how to shuffle parallel implementations? *International Journal of Circuit Theory and Applications*, 48(5):674–695, 2020.

[LCWY09] Huiyun Li, Tingding Chen, Keke Wu, and Fengqi Yu. Quantitative evaluation of side-channel security. In *2009 Asia-Pacific Conference on Information Processing*, volume 2, pages 456–460. IEEE, 2009.

[LMR22] Pascal Lafourcade, Gael Marcadet, and Léo Robert. Faster non-interactive verifiable computing. *IACR Cryptol. ePrint Arch.*, page 646, 2022.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 20–33. Springer, 2015.

[LPS08] Yehuda Lindell, Benny Pinkas, and Nigel P Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *International Conference on Security and Cryptography for Networks*, pages 2–20. Springer, 2008.

[Man04] Stefan Mangard. Hardware countermeasures against dpa–a statistical analysis of their effectiveness. In *Cryptographers' Track at the RSA Conference*, pages 222–235. Springer, 2004.

[MS16] Amir Moradi and François-Xavier Standaert. Moments-correlating dpa. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, pages 5–15, 2016.

[MTH+21] Shayan Moini, Shanquan Tian, Daniel Holcomb, Jakub Szefer, and Russell Tessier. Power side-channel attacks on bnn accelerators in remote fpgas. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(2):357–370, 2021.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *EC*, pages 129–139, 1999.

[PKH+21] Dongjun Park, GyuSang Kim, Donghoe Heo, Suhri Kim, HeeSeok Kim, and Seokhie Hong. Single trace side-channel attack on key reconciliation in quantum key distribution system and its efficient countermeasures. *ICT Express*, 7(1):36–40, 2021.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 513–533. Springer, 2017.

[PR10] Emmanuel Prouff and Matthieu Rivain. Theoretical and practical aspects of mutual information-based side channel analysis. *International Journal of Applied Cryptography*, 2(2):121–138, 2010.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *ASIACRYPT*, pages 250–267, 2009.

[PZS17] Romain Poussier, Yuanyuan Zhou, and François-Xavier Standaert. A systematic approach to the side-channel analysis of ecc implementations with worst-case horizontal attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 534–554. Springer, 2017.

[RPD+18] Chethan Ramesh, Shivukumar B Patil, Siva Nishok Dhanuskodi, George Provelengios, Sébastien Pillement, Daniel Holcomb, and Russell Tessier. Fpga side channel attacks without physical access. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 45–52. IEEE, 2018.

[SGMT18] Falk Schellenberg, Dennis RE Gnad, Amir Moradi, and Mehdi B Tahoori. Remote inter-chip power analysis side-channel attacks at board-level. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2018.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 30–46. Springer, 2005.

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

[SZ13] Thomas Schneider and Michael Zohner. Gmw vs. yao? efficient secure two-party computation with low depth circuits. In *International Conference on Financial Cryptography and Data Security*, pages 275–292. Springer, 2013.

[TBP20] Florian Tramèr, Dan Boneh, and Kenny Paterson. Remote {Side-Channel} attacks on anonymous transactions. In *29th USENIX security symposium (USENIX security 20)*, pages 2739–2756, 2020.

[TMC+21] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Cagdas Calik, Lawrence Bassham, Jinkeon Kang, John Kelsey, et al. Status report on the second round of the nist lightweight cryptography standardization process. *National Institute of Standards and Technology Internal Report*, 8369(10.6028), 2021.

[TSR+20] Eleonora Testa, Mathias Soeken, Heinz Riener, Luca Amaru, and Giovanni De Micheli. A logic synthesis toolbox for reducing the multiplicative complexity in logic networks. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 568–573. IEEE, 2020.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 740–757. Springer, 2012.

[vWWB11] Jasper GJ van Woudenberg, Marc F Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In *Cryptographers' Track at the RSA Conference*, pages 104–119. Springer, 2011.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[YWZ20] Kang Yang, Xiao Wang, and Jiang Zhang. More efficient MPC from improved triple generation and authenticated garbling. In *CCS*, pages 1627–1646, 2020.

[ZCD+19] Lianying Zhao, Joseph I. Choi, Didem Demirag, Kevin R. B. Butler, Mohammad Mannan, Erman Ayday, and Jeremy Clark. One-time programs made practical. In *FC*, pages 646–666, 2019.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, pages 220–250, 2015.