Attacks and Countermeasures for White-box Designs

Alex Biryukov, Aleksei Udovenko

`first-name.last-name@uni.lu`
SnT, University of Luxembourg

Abstract. In the traditional symmetric cryptography, the adversary has access only to the inputs and outputs of a cryptographic primitive. In the *white-box* model the adversary is given full access to the implementation. He can use both static and dynamic analysis as well as fault analysis in order to break the cryptosystem, e.g. to extract embedded secret key. Implementations secure in such model have many applications in industry. However, creating such implementations turns out to be a very challenging if not an impossible task.
Recently, Bos *et al.* [5] proposed a generic attack on white-box primitives called differential computation analysis (DCA). This attack applies to most existent white-box implementations both from academia and industry. The attack comes from side-channel cryptanalysis method. The most common method protecting against such side-channel attacks is masking. Therefore, masking can be used in white-box implementations to protect against the DCA attack. In this paper we investigate this possibility and present multiple generic attacks against masked white-box implementations. We use the term "masking" in a very broad sense. As a result, we deduce new constraints that any secure white-box implementation must satisfy. We suggest partial countermeasures against the attacks.
Some of our attacks were successfully applied to the WhibOx 2017 [13] challenges.

Keywords: White-box, obfuscation, cryptanalysis

1 Introduction

White-box cryptography aims to develop cryptographic primitives that can withstand attacks of very powerful adversaries. Those adversaries have full access to the implementations, either in the form of source code or in the form of compiled programs. They can perform both static and dynamic analysis, including debugging, tracing the execution, injecting faults, modifying the program parts, etc. Cryptographic implementation resistant to such attacks is also called *strong white-box* since it is essentially equivalent to a public key scheme achieved by code-obfuscation means.

In 2002, Chow *et al.* [10,11] proposed the first white-box implementations of the AES and DES block ciphers. The main idea was to represent small parts of a block cipher as look-up tables and compose them with randomized invertible mappings to hide the secret key information. Each such look-up table by itself does not give any information about the key. In order to attack such scheme, multiple tables must be considered. Another approach was proposed by Bringer *et al.* [8]. Instead of look-up tables, the cipher is represented as a sequence of functions over $GF(2^n)$ for some n, with some additional computations as noise. These functions are then composed with random linear mappings to hide the secret key, similarly to the Chow *et al.* approach.

Unfortunately, both approaches fell to practical attacks [1, 12, 21]. Consequent attempts to fix them were not successful [20, 22, 24]. Moreover, Michiels *et al.* generalized attack by Billet *et al.* [1] and showed that Chow *et al.* approach is not secure for any SPN cipher with MDS matrices. This follows from the efficient cryptanalysis of any SASAS structure [4]. Recently several white-box schemes based on ASASA structure were proposed [2]. However the strong white-box scheme from that paper was broken [3, 15, 23] (which also broadens the white-box attacker's arsenal even further). Another recent approach consists in obfuscating a block cipher implementation using candidates for indistinguishability obfuscation (e.g. [14]).

Besides academia, there are commercial white-box solutions which are used in real products. The design behind those implementations is kept secret, thus adding *security-by-obscurity* protection. Nevertheless, Bos *et al.* [5] proposed a framework for attacks on white-box implementations, which can automatically break many white-box implementations. The idea is to apply techniques from grey-box analysis (i.e. side-channel attacks) but using more precise data traces obtained from the implementation. The attack is called *differential computation analysis* (DCA).

In light of such powerful automated attack the question arises: how to create a white-box scheme secure against the DCA attack? The most common countermeasure against side-channel attacks is masking. It is natural therefore to apply masking to protect white-box implementations. We define masking to be any obfuscation method that encodes each original bit by a relatively small amount of bits. Such masking-based obfuscation may be more practical in contrast to cryptographic obfuscation built from current indistinguishability obfuscation candidates [9, 14]. Even if the performance improvement comes at the cost of less rigorous security analysis.

In this paper we describe multiple generic attacks against masked implementations and conclude that the classic Boolean masking (XOR-sharing) is inherently weak. We summarize previous and new attacks in Table 1. More general nonlinear encodings are needed and we deduce constraints from our attacks. We believe that our results provide new insights on design of white-box implementations. We suggest three nonlinear masking constructions as countermeasures for the attacks. Together with classic linear masking scheme these constructions solve the "value hiding" problem (see Section 4). The only missing component is a "structure hiding" obfuscation which has to hide only the structure of the circuit. Development of such obfuscation methods is left as a future work.

Outline We provide the notations in Section 2. The general attack setting and the attacks are described in Section 3. We discuss possible countermeasures in Section 4. Finally, we conclude and suggest future work in Section 5.

2 Notations and Definitions

Throughout the paper, we use the following notations and definitions.

- $\wedge, \vee, \oplus$ denote Boolean AND,OR and XOR respectively.
- $exp_a(b)$ stands for a^b.
- $\mathbb{F}_2$ is the finite field of size 2 and $\mathbb{F}_2^n$ is the vector space over $\mathbb{F}_2$ of dimension n.
- Elements in vectors are indexed starting from 1. For a vector v from $\mathbb{F}_2^n$ we write $v = (v_1, \ldots, v_n)$.

Table 1: Attacks on masked white-box implementations.

Attack	Ref.	Data	Time
Correlation	[5],Sec. 3.1	$\mathcal{O}(2^t)$	$\mathcal{O}(n^t k 2^{2t})$
Time-Memory Tradeoff	Sec. 3.1	$\mathcal{O}(1)$	$\mathcal{O}(n^{\lceil s/2 \rceil} + n^{\lfloor s/2 \rfloor} k)$
Linear Algebra	[16],Sec. 3.2	$\mathcal{O}(n)$	$\mathcal{O}(n^\omega + n^2 k)$
Generalized Lin. Alg.	Sec. 3.2	$\mathcal{O}(\sigma(n,d)))$	$\mathcal{O}(\sigma(n,d)^\omega + \sigma(n,d)^2 k)$
2-Share Fault Injection	Sec. 3.3	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
1-Share Fault Injection	Sec. 3.3	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$

Notations: n denotes size of the obfuscated circuit or its part selected for the attack; s is the number of shares in the masking scheme; k is the number of key candidates required to compute a particular intermediate value in the circuit; t denotes the correlation order ($t \leq s$); ω is the matrix multiplication exponent (e.g. $\omega = 2.8074$ for Strassen algorithm); d is the algebraic degree of the masking decoder (see Section 3); $\sigma(n,d) = \sum_{i=0}^{d} \binom{n}{i}$ is the number of monomials of n bit variables of degree at most d .

- $|X|$ denotes the size of the vector/set X. If X is a circuit, $|X|$ denotes number of nodes in it.
- *Weight* of a vector v is the number or nonzero entries in it and is denoted $wt(v)$.
- *Correlation* of two n-bit vectors v_1 and v_2 is defined as $cor(v_1, v_2) = 1 - 2 \cdot wt(v_1 \oplus v_2)/n$.
- Any Boolean function f with n-bit input has unique representation of the form $f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u x^u$ called algebraic normal form (ANF). Here x^u is a shorthand for $x_1^{u_1} \ldots x_n^{u_n}$ and such products are called *monomials.*
- The algebraic degree of a Boolean function is the maximum Hamming weight of all u such that $a_u = 1$. Equivalently, it is the maximum degree of a monomial in the ANF of f.
- $\sigma(n,d) = \sum_{i=0}^{d} \binom{n}{i}$ is the number of monomials of n bit variables from $\mathbb{F}_2$ of degree at most d.
- *Restriction* of a Boolean function f by predicates $p_1, p_2, \ldots$ is denoted by $f\langle p_1, p_2, \ldots \rangle$. It is equal to the product of the functions: $f\langle p_1, p_2, \ldots \rangle(x) = f(x) \cdot p_1(x) \cdot p_2(x) \cdot \ldots$.
- The closure of a set S under addition is denoted by $span(S)$.

3 Attacks On Masked Implementations

We describe the general setting for our attacks. We consider *a keyed symmetric primitive*, e.g. a block cipher. The *white-box designer* takes a naive implementation with a hardcoded secret key and obfuscates it producing *a white-box implementation.* An adversary receives the white-box implementation and her goal is to recover the secret key or a part of it. We restrict our analysis to implementations in the form of *Boolean circuits.*

Masking. We assume that the white-box designer uses masking in some form, but we do not restrict him from using other obfuscation techniques. The only requirement is that there exists a relatively small set of nodes in the obfuscated circuit (called *shares*) such that during a legitimate computation the values computed in these nodes sum to *a predictable value*. We at least expect this to happen with overwhelming probability. In a more general case, we allow arbitrary functions to be used to compute the predictable value from the shares instead of plain XOR. We call these functions *decoders*. The classic Boolean masking technique is based on the XOR decoder. The number of shares is denoted by s.

A predictable value typically is a value computed in the beginning or in the end of the reference algorithm such that it depends only on a few key bits and on the plaintexts/ciphertexts. In such case the adversary makes a guess for the key bits and computes the corresponding candidate for the predictable value. The total number of candidates is denoted by k.

The obfuscation method may require random bits e.g. for splitting the secret value into random shares. Even if the circuit may have input nodes for random bits in order to achieve non-deterministic encryption, the adversary can easily manipulate them. Therefore, the obfuscation method has to rely on pseudorandomness computed solely from the input. We assume that the adversary can not directly locate the pseudorandomness computations and remove the corresponding nodes. Moreover, the adversary can not predict the generated pseudorandom values with high probability, i.e. such values are not predictable values.

Window coverage. In a typical case shares of a predictable value will be relatively close in the circuit. This fact can be exploited to improve efficiency of the attacks. We cover the circuit by sets of closely located nodes. Any such set is called *a window* (as in power analysis attack terminology e.g. from [6]). The described attacks can be applied to each window instead of the full circuit. By varying the window size the attacks may become more efficient. Here we do not investigate methods of choosing the windows to cover a given circuit. One possible approach is to assign each level or a sequence of adjacent levels in the circuit to a window. Choosing the full circuit as a single window is also allowed. In our attacks we assume that a coverage is already chosen. For simplicity, we describe how each attack is applied to a single window. In case when multiple windows are chosen, the attack has to be repeated for each window. The window size is denoted by n. It is equal to the circuit size in case of a single window coverage.

General DCA attack. We would like to note that the term “differential computation analysis” (DCA) is very general. In [5] the authors introduced it mainly for the correlation-based attack. In fact our new attacks fit the term well and provide new tools for the “analysis” stage of the attack. The first stage remains the same except that we adopt the terminology for the case of Boolean circuits instead of recording the memory access traces. Our view of the procedure of the DCA attack on a white-box implementation C is given in the Algorithm 1.

We remark that the correlation-based DCA attack from [5] can be implemented on-the-fly, without computing the full vectors v_j. In contrast, most of our attacks require the full vectors.

In the following two sections we describe two classes of DCA attacks: combinatorial and algebraic. They both follow the procedure described above and differ only in the analysis part (Step 6a). Afterwards, we describe two fault-injection attacks which allow to find locations of shares efficiently.

Algorithm 1 General procedure of DCA attacks on Boolean circuits.

```
1. Generate a random tuple of plaintexts $P = (p_1, p_2, \ldots)$.
2. For each plaintext $p_i$ from $P$:
   (a) Compute the circuit $C$ on input $p_i$: $c_i = C(p_i)$.
   (b) For each node indexed $j$ in the circuit:
       i. Record the computed value as $v_{j,i}$.
   (c) For each predictable value indexed $j$:
       i. Record the predictable value computed using plaintext $p_i$ (or ciphertext $c_i$) as $\tilde{v}_{j,i}$.
3. Generate a bit-vector $v_j = (v_{j,1}, \ldots, v_{j,|P|})$ of computed bits for each node $j$ in the circuit.
   Denote by $V$ the set of all vectors $v_j$: $V = \{v_1, \ldots, v_{|C|}\}$.
4. Generate a bit-vector $\tilde{v}_j = (\tilde{v}_{j,1}, \ldots, \tilde{v}_{j,|P|})$ for each predictable value $j$. Denote by $\tilde{V}$ the set
   of all predictable vectors $\tilde{v}_j$: $\tilde{V} = \{\tilde{v}_1, \ldots, \tilde{v}_k\}$.
5. Choose a coverage of $V$ by windows of size $n$.
6. For each window in the coverage:
   (a) Perform analysis on the window $W \subseteq V$ using the set of predictable values $\tilde{V}$.
```

3.1 Combinatorial DCA attacks

The most straightforward way to attack a masked implementation is to guess location of shares inside the current window. For each guess we need to check if the shares match the predictable value. In the basic case of classic Boolean masking where the decoder function is simply XOR of the shares the check is trivial. If an unknown general decoder function has to be considered, the attack becomes more difficult. One particularly interesting case is a basic XOR decoder with added noise. The main attack method in such cases is correlation.

Correlation attack. The correlation DCA attack from [5] is based on correlation between single bits. However, in the case of classic Boolean masking with strong pseudorandom masks all s shares are required to perform a successful correlation attack. In the case of a nonlinear decoder less shares may be enough: even a single share correlation can break many schemes as demonstrated in [5]. Existing higher-order power analysis attacks are directly applicable to memory or value traces of white-box implementations. However, the values leaked in the white-box setting are exact in contrast to side-channel setting and the attack may be described in a simpler way. We reformulate the higher-order correlation attack in our DCA framework.

Assume that locations of t shares are guessed and t vectors v_j are selected. For simplicity, we denote them by $(v_1, \ldots, v_t)$. For each t-bit vector m we compute the bit-vector u_m where

$$u_{m,i} = (v_{1,i} = m_1) \wedge \ldots \wedge (v_{t,i} = m_t).$$

In other words, $u_{m,i}$ is equal to 1 if and only if during encryption of the i-th plaintext the shares took the value described by m . For each predictable vector $\tilde{v}$ we compute the correlation $cor(u_m, \tilde{v})$. If its absolute value is above a predefined threshold, we conclude that the attack succeeded and possibly recover part of the key from the predictable value $\tilde{v}$. Furthermore, the entire vector of correlations $(cor(u_{(0,\ldots,0)}, \tilde{v}), cor(u_{(0,\ldots,1)}, \tilde{v}), \ldots)$ may be used in analysis, e.g. the average value of its absolute entries.

For the attack to succeed we need the correlated shares to hit at least one combination m constant number of times (that is obtain $wt(u_m) \geq const$). Therefore the data complexity is $|P| = \mathcal{O}(2^t)$. However, with larger number of shares the noise increases and more data may be required. We estimate the time complexity of the attack as $\mathcal{O}(n^t k 2^t |P|) = \mathcal{O}(n^t k 2^{2t})$. Here n^t corresponds to guessing location of shares inside each window (we assume $t \ll n$); k corresponds to iterating over all predictable values; 2^{2t} corresponds to iterating over all t-bit vectors m and computing the correlations.

The main advantage of this attack is its generality. It works against general decoder functions even with additional observable noise. In fact, the attack may work even if we correlate less shares than the actual encoding requires. Indeed, the attack from [5] relied on single-bit correlations and still was successfully applied to break a lot of whitebox designs. The generality of the attack makes it inefficient for some special cases, in particular for the classic Boolean masking. We investigate this special case and describe more efficient attacks.

Time-Memory Trade-off. We now consider the case of XOR decoder and absence of observable noise. That is, the decoder function must map the shares to the correct predictable value for all recorded plaintexts. In such case we can use extra memory to improve the attack. Consider two simple cases by number of shares:

1. Assume that the decoder uses a single share (i.e. unprotected implementation). We precompute all the predictable vectors and put them in a table. Then we simply sweep through the circuit nodes and for each vector v_i check if it is in the table. For the right predictable vector $\tilde{v}$ we will have a match.
2. Assume that the decoder uses two shares (i.e. first-order protected implementation). We are looking for indices i, j such that $v_i \oplus v_j = \tilde{v}$ for some predictable vector $\tilde{v}$. Equivalently, $v_i = \tilde{v} \oplus v_j$. We sweep through the window's nodes and put all the node vectors in a table. Then we sweep again and for each vector v_j in the window and for each predictable vector $\tilde{v}$ we check if $v_j \oplus \tilde{v}$ is in the table. For the right $\tilde{v}$ we will have a match and it will reveal both shares.

This method easily generalizes for arbitrary number of shares. We put the larger half of shares on the left side of the equation and put the corresponding tuples of vectors in the table. Then we compute the tuples of vectors for the smaller half of shares and look-up them in the table. We remark that this attack's complexity still has combinatorial explosion. However the time-memory trade-off essentially allows to half the exponent in the complexity.

The attack effectively checks $n^s k$ sums of vectors to be equal to zero. To avoid false positives, the data complexity should be set to $O(s \log_2 n + \log_2 k)$. We consider this data complexity negligible, especially because for large number of shares the attack quickly becomes infeasible. The attack becomes irrelevant for large numbers of shares. For simplicity, we assume the data complexity is $O(1)$. Then, the time complexity of the attack is $\mathcal{O}((n^{\lceil s/2 \rceil} + n^{\lfloor s/2 \rfloor} k))$.

The described attack is very efficient for unprotected or first-order masked implementations. For small windows it can also be practical for higher-order protections. In the following section we describe a more powerful attack whose complexity is independent of number of shares.

3.2 Algebraic DCA attacks

For the classic Boolean masking the problem of finding shares consists in finding a subset of the window's vectors which sums to the predictable value. Clearly, this is a basic linear algebra problem. Let M be the matrix whose columns are vectors from the current window. For each predictable vector $\tilde{v}$ we solve the equation $M \times x = \tilde{v}$. Any solution vector x reveals shares locations. To avoid false-positive solutions the number $|P|$ of encryptions should be increased proportionally to the window size. For the same matrix M we need to check all predictable vectors. Instead of solving the entire system each time, we precompute the LU decomposition of the matrix and then use it for checking each predictable vector much faster. We estimate the data complexity $|P| = \mathcal{O}(n)$ and the time complexity $\mathcal{O}(n^\omega + n^2 k)$, where ω is the matrix multiplication exponent. This attack was independently discovered by the CryptoExperts team and among other techniques was successfully applied [16] during the WhibOx 2017 competition [13] in order to break the winning challenge "Adoring Poitras".

We remark that the described attack does not necessary find a solution with minimal possible number of shares. In general, finding a low-weight solution of a linear equation system is known to be a hard problem. Nevertheless, the attack still allows to find the correct predictable vector. The discovered locations of shares also provide valuable information about the white-box design.

We conclude that classic Boolean masking is insecure regardless of the number of shares. The attack complexity is polynomial in the circuit size. Even though it may not be highly practical to apply the attack to entire circuits containing millions of nodes, good window coverage makes the attack much more efficient. The attack becomes especially dangerous if a window containing all shares may be located by analyzing the circuit. Indeed, this is how team CryptoExperts attacked the main circuit of the winning challenge of the WhibOx competition. They obtained a minimized circuit containing around 300000 nodes; they draw the data dependency graph (DDG) of the top 5% nodes and visually located several groups of 50 nodes and successfully mounted the described linear attack on each of the groups.

Generalization through linearization. The described linear attack suggests that a nonlinear masking scheme has to be used. We show that the attack can be generalized to nonlinear masking schemes as well. Of course, the complexity grows faster. Still, the attack can be used to estimate the security of such implementations.

The generalization is based on the linearization technique. The idea is to compute products of vectors (with bitwise AND) and include them as possible shares of the predictable vector. Each such product corresponds to a possible monomial in the algebraic normal form of the decoder function. The correct linear combination of monomials equals to the decoder function. The corresponding linear combination of products of vectors equals to the correct predictable vector.

The set of products may be filtered. If a bound on the degree of the decoder function is known, products with higher degrees are not included. For example, for a quadratic decoder function only the vectors v_i and all pairwise products $v_i v_j$ should be included.

The data complexity is dependent on the number of possible monomials in the decoder function. For simplicity, we consider an upper bound d on the algebraic degree. Then the number of possible monomials is equal to $\sigma(n,d) = \sum_{i=0}^{d} \binom{n}{i}$. This generalized attack has the data complexity $\mathcal{O}(\sigma(n,d))$ and the time complexity $\mathcal{O}(\sigma(n,d)^\omega + \sigma(n,d)^2 k)$.

We describe an interesting scenario where this generalized attack is highly relevant. Assume that a white-box designer first applies classic Boolean masking to the reference circuit. Afterwards, each intermediate bit is encoded by e.g. 8 bits using a random non-linear encoding. The masked circuit then is transformed into a network of lookup tables which perform operations on the encoded bits without explicitly decoding them. The motivation for such scheme is that there will be no correlation between a single 8-bit encoding and any predictable vector because of the linear masking applied under the hood. For the generalized linear attack the degree bound is equal to 8 and normally, the time complexity would be impractical. However, in this case the lookup tables reveal the locations of encodings, i.e. the 8-bit groups. Therefore, we include only all 2^8 products from each group and no products between groups. The attack works because the predictable value is a linear combination of XOR-shares which in turn are linear combinations of products (monomials) from each group.

3.3 Fault Attacks

Initially, we assumed that the adversary knows the obfuscated circuit and can analyze it in an arbitrary way. Still, the attacks described in previous sections were passive: they relied on analysis of computed intermediate values during encryptions of random plaintexts. In this section we show that active attacks (fault injections) can also be used to attack masked white-box implementations. We assume that the classic Boolean masking is used. We also allow any form of integrity protection which protects the values but does not protect the shares.

Two-Share Fault Injection. The main goal of a fault attack against masking is to locate shares of the masked values. Observe that flipping two XOR-shares of a value does not change the value. The attack follows:

1. Encrypt a random plaintext p and save the ciphertext $E(p)$.
2. Choose two intermediate nodes c_i, c_j and flip them during encryption of p. Denote the ciphertext by $E'(p)$.
3. If $E(p) = E'(p)$, conclude that c_i, c_j are shares of the same value (possibly repeat check for other plaintexts). Otherwise try another two intermediate nodes.

As shares of the same value should be placed closely in the circuit, a window coverage can be used to improve efficiency of this attack too. The idea is to choose two shares only inside each window and not across the windows.

The described attack allows to locate all shares of each value, independently of the sharing degree. The attack performs $\mathcal{O}(n^2)$ encryptions and has time complexity $\mathcal{O}(|C|n^2)$.

One-Share Fault Injection. Recall that we allow an integrity protection on the values but not on the shares. One possible way an integrity protection may be implemented is to perform the computations twice and spread the difference between two results across the output in some deterministic way. In such way small errors are amplified into random ciphertext differences. In case of such protection or absence of any protection we can improve the efficiency of the fault attack.

The main idea for improvements comes from the following observation: if we flip any single share of the same value, the masked value will be flipped as well. This results in a fault injected in the unmasked circuit. Note that the circuit output does not depend on which share was faulted. This observation allows to split the two-share fault attack and perform fault injection only for each node instead of each pair of nodes, at the cost of additional storage:

1. Encrypt a random plaintext p and save the ciphertext $E(p)$.
2. For each intermediate node c_i:
   (a) Flip the value of c_i during encryption of p. Denote the ciphertext by $E'_i(p)$.
   (b) Store $E'_i(p)$ in a table.
   (c) If $E'_i(p)$ was already stored in the table as $E'_j(p)$ we learn that nodes c_i and c_j are shares of the same value.

The attack performs $\mathcal{O}(n)$ encryptions, which requires $\mathcal{O}(|C|n)$ time. It provides substantial improvement over previous attack, though it requires stronger assumption about the implementation. The most relevant counter-example is when the integrity protection does not amplify the error but simply returns a fixed output for any detected error. In a sense, such protection does not reveal any information about the fault in the output. On the other hand, it may be possible to locate the error checking part in the circuit and remove the protection.

The attacks can be adopted for nonlinear masking as well. In such case the injected fault may leave the masked value unflipped. When a zero difference is observed in the output, the fault injection should be repeated for another plaintexts. As plaintext is the only source of pseudorandomness, changing the plaintext should result in different values of shares. Flipping a share would result in flipping the masked value with nonzero probability. The exact probability depends on the decoder function.

Remark. The two described attacks perform faults on *nodes* of the circuit. In some cases, a node value may be used as a share of multiple different values. For example, if the same pseudorandom value is used to mask several values. A more general variant of attacks would inject faults on *wires*. However, multiple wires have to be faulted in order to have a guaranteed fault of a share. Though, the guaranteed fault is not necessary for the attack. Intuitively, getting the same faulted output by flipping different nodes or wires uncovers important structural information about the white-box design (if the space of faulted outputs is large enough).

4 Countermeasures

The attacks described in the previous section significantly narrow down the space of masking schemes useful for white-box obfuscation. We deduce the following main constraints:

1. The number of shares must be high enough to avoid combinatorial attacks. Moreover, the minimum number of shares that correlate with reference circuit values must be high as well.
2. The decoder functions must have high enough degree to avoid algebraic attack.
3. The circuit must not admit analysis which allows to locate shares of the same values.
4. The integrity of pseudorandom shares must be protected.

The aim of this paper is to analyze the possibility of using masking schemes with *relatively small* number of shares for the white-box cryptography. The complexity of combinatorial attacks splits into two parts: locating the shares and correlating them. If the number of shares is very high then the correlation part becomes infeasible. Possibly, in such case it is not even necessary to hide the location of shares. The downside is that designing such masking schemes is quite challenging and this direction leads into rather theoretical constructions like indistinguishability obfuscation [14] from fully homomorphic encryption and other cryptographic primitives. We aim to find more practical obfuscation techniques. Therefore, we study obfuscation methods relying on hardness of locating shares inside the obfuscated circuit. Such obfuscation is a challenging problem. In the light of described attacks, we split the problem into two components:

1. *(Value Hiding)* Protection against generic passive attacks which do not rely on the analysis of the circuit.
2. *(Structure Hiding)* Protection against circuit analysis and fault injections.

Value Hiding. The first component basically requires designing a proper masking scheme. As we have shown, the requirements are much stronger than for the usual masking in the side-channel setting (e.g. the provably secure masking by Ishai *et al.* [19]). To the best of our knowledge, this direction was not studied in the literature. However, there is a closely related notion: fully homomorphic encryption (FHE). Indeed, it can be seen as an extreme class of masking schemes. FHE encryption is a process of creating shares of a secret value and the FHE's evaluation functions allow to perform arbitrary computations on the ciphertexts (shares) without leaking the secret value. In fact, any secure FHE scheme would solve the "Value Hiding" problem. However, this direction leads to very inefficient schemes: typically FHE schemes have very large ciphertexts. This contradicts our goal to investigate schemes with small number of shares.

We suggest to further split the first component into two parts. The first part is protection against the algebraic attacks. It is a nonlinear masking scheme without low-degree decoders. However, we allow the scheme to be imperfect: the computed values may correlate with the secret values. The second part is protection against correlation attacks. It can be implemented using a provably secure linear masking scheme on top of the nonlinear masking from the first part. The two parts may be composed in the following way: the algebraically secure nonlinear masking scheme is applied to the reference circuit and afterwards the linear masking scheme is applied to the transformed circuit. We investigate possibilities for the algebraically secure nonlinear masking in the next section.

Structure Hiding The second component resembles what is usually understood by *software obfuscation*. Indeed, the usual software obfuscation aims to obfuscate the control flow graph and hide important operations. Often such obfuscation includes integrity protections to avoid patching. The computed values are not hidden but merely blended among redundant values computed by dummy instructions. For circuits the problem is less obscure and ad hoc. In particular, an integrity protection scheme for circuits was proposed by Ishai *et al.* in [18]. Though, formalizing the "protection against analysis" is not trivial. Applying structure hiding protection on top of value hiding protection should secure the implementation from attacks described in Section 3. We do not investigate structure hiding further in this paper and leave it as future work.

4.1 Algebraically Secure Masking Schemes

We give a broad definition of a masking scheme which we will use throughout this section.

Definition 1 (Masking Scheme). *An N-bit masking scheme is defined by a non-constant decoding function $Dec : \mathbb{F}_2^N \to \mathbb{F}_2$ and a set of triplets $\{(\diamond, Eval_\diamond, C_\diamond), \ldots\}$ where each triplet consists of:*

1. *a Boolean operator $\diamond_i : \mathbb{F}_2 \times \mathbb{F}_2 \to \mathbb{F}_2$,*
2. *a function $Eval_\diamond : \mathbb{F}_2^N \times \mathbb{F}_2^N \times \mathbb{F}_2^R \to \mathbb{F}_2^N$,*
3. *a circuit $C_\diamond$ computing $Eval_\diamond$.*

The following equation must be satisfied for all operators $\diamond_i$ and all values r:

$$Dec(Eval_\diamond(x_1, x_2, r)) = Dec(x_1) \diamond Dec(x_2).$$

The degree of the masking scheme is the algebraic degree of the Dec function. The masking scheme is called nonlinear if its degree is at least equal to 2. It is allowed to use different decoding functions for two inputs and output.

Note that $Eval_\diamond$ takes three arguments in our definition. The first two are masks of the secret values and the third one is optional randomness which must not have effect on the secret values.

We now define the algebraic security of a masking scheme. Note that in a reference circuit of a cryptographic primitive two nodes may be used as AND operands and as XOR operands simultaneously. The algebraic attack then would include combinations of nodes across the two masked operations circuits. Therefore, we need to analyze all $Eval_\diamond$ circuits together.

Definition 2 (Algebraic Security). *Consider an N-bit masking scheme defined as above. Let $\mathcal{F}$ be a set of functions computed in the nodes of all circuits $C_\diamond$. Let $\mathcal{F}^d$ be a set of functions obtained by applying all Boolean functions of degree at most d to functions from $\mathcal{F}$. Furthermore, let $f[\tilde{x}, \tilde{y}]$ be the restriction $f\langle Dec(x) = \tilde{x}, Dec(y) = \tilde{y}\rangle$. The masking scheme is called d-th order algebraically secure if for any non-constant function f from $\mathcal{F}^d$ all its restrictions $f[\tilde{x}, \tilde{y}]$ are non-constant.*

The definition captures only protection against the algebraic attack and only for a single masked operation. Composing masked operations may require additional randomness for refreshing the masks. Given a masking scheme, it is possible to evaluate its algebraic security by running the algebraic attack on the joined $Eval_\diamond$ circuits. For higher degrees of protection the attack may become impractical even if applied directly to the $Eval_\diamond$ circuits. Therefore, a provably secure construction for arbitrary orders is an important challenge. Still, the first important problem is how to protect a scheme against the *linear* algebraic attack. Indeed, even second degree generalized attack may be impractical for large circuits. Therefore, we first aim to find the simplest nonlinear masking scheme.

Minimalist Quadratic Masking. Since the decoding function has to be at least quadratic, we need at least two bits to encode a single bit. For two bits all nonlinear decoding functions are linear equivalent to a quadratic monomial being simply product of the two input bits. Unfortunately, this decoding function is vulnerable to a variant of the linear algebra attack. Any quadratic function with 2-bit input is unbalanced. Therefore, one of the reference bit values can be encoded by 3 different values and the other value has only 1 possible encoding. For example, if the value is equal to 1 and the decoding function is simply AND, the input has to be equal to (1, 1). In this case there is no randomness involved. The adversary may filter plaintexts and choose only those for which the predictable value of the reference node is equal to 1. Then she will know that the masks are equal to 1 in this subset of executions. She can not apply the linear algebra attack directly, because the all-one vector would typically occur in the circuit in many places. However, she can add a single plaintext that is not in the subset. In this case one of the shares has to be equal to zero and the share's vector becomes all-one vector followed by one zero. Such vector would not occur in the circuit by chance because the subset of plaintexts was specially crafted. This attack has to be taken into consideration when designing a nonlinear masking scheme. The conclusion is that any value of the original bit should include randomness in its encoding. In particular, the decoding function can not be a point function. That's why the definition of algebraic security uses restrictions of the functions.

As we showed that there does not exist secure 2-bit encoding, we move on to 3-bit encodings. The simplest quadratic function using all 3 input bits a, b, c is $ab \oplus c$. Note the similarity with the broken 2-bit scheme: the quadratic monomial ab is simply linearly masked by c. However, this linear mask is enough to prevent the attack: in this case $Dec(a, b, c) = 1$ does not imply $a = 1$ or $b = 1$. In fact, such Dec is balanced: both 0 and 1 have exactly 4 preimages. The expressions for masked operations are not hard to find as well. In addition, we also describe a refresh-gadget which injects external randomness into mask:

Fig. 1: Minimalist Quadratic Masking Scheme.

$$
\begin{aligned}
Dec(a,b,c) &= ab \oplus c, &(1)\\
Eval_{XOR}((a,b,c),(d,e,f)) &= (a \oplus d,\ b \oplus e,\ ae \oplus bd \oplus c \oplus f), &(2)\\
Eval_{AND}((a,b,c),(d,e,f)) &= (ae,\ bd,\ (cd)e \oplus a(bf) \oplus cf), &(3)\\
Refresh((a,b,c),(r_a,r_b)) &= (a \oplus r_a,\ b \oplus r_b,\ c \oplus r_a b \oplus r_b a \oplus r_a r_b). &(4)
\end{aligned}
$$

It is easy to verify that $Eval_{XOR}$ and $Eval_{AND}$ satisfy the requirements from the Definition 1. In addition, $Refresh(a, r)$ returns fresh random encoding of $Dec(a)$, meaning that $Dec(a) = Dec(Refresh(a, r))$ for any r and new masks reveal no information about the old masks. Extending the $Eval$ functions by refreshing input and output masks with fresh random values makes each application of $Eval$ independent from the others and allows free composition of the masked operations.

The proposed masking scheme is first-order algebraically secure for circuits based on the provided expressions (the order of XORs is not important). The space of all linear combinations of functions computed in the circuit nodes is spanned by the computed monomials $a, b, c, d, e, f, ae, bd, cd, cde, bf, abf, cf$. Note that a restriction of a linear combination of monomials is equal to the linear combination of restrictions of monomials.

Therefore, we only need to check that there are no constant linear combinations of restrictions of the monomials by predicates $(ab \oplus c = \tilde{x}, de \oplus f = \tilde{y})$. We confirmed this and also experimentally verified that the linear algebra attack does not succeed.

Random Encodings. In search of higher-order secure schemes we investigate the possibility of using random nonlinear encodings. The idea is to use random decoding functions for each bit in the circuit. Each $Eval_\diamond$ each takes two bits with decoders Dec_1, Dec_2 and returns bit with a random decoder Dec. The function $Eval_\diamond$ is constructed by defining

$$Eval_\diamond(x_1, x_2) = Enc\big(Dec_1(x_1) \diamond Dec_2(x_2), r_1(x_1, x_2), \ldots, r_{N-1}(x_1, x_2)\big),$$

where $r_1, \ldots, r_{N-1}$ are some random Boolean functions generating the output randomness from the input shares and Enc is a random N-bit bijection. Observe that Enc^{-1} contains the output decoder: the first bit of $Enc^{-1}(Eval_\diamond(x_1, x_2))$ is equal to the result of unmasked operation $Dec_1(x_1) \diamond Dec_2(x_2)$. Finally, $Eval_\diamond$ is implemented as a circuit *from its look-up table* using some generic method to avoid computation of unmasked values.

Unfortunately, random encodings are unlikely to completely solve the problem. We describe a heuristic argument which shows that circuits implementing masked operations on random encodings are likely to be susceptible to the algebraic attacks. The argument is based on counting functions. First, we show that number of nonlinear gates in the circuit implementing masked operation $Eval_\diamond$ is a relevant parameter. Circuits with fewer nonlinear gates have more chances to be secure against the algebraic attack.

Proposition 1. *Consider an arbitrary Boolean circuit. Let $\mathcal{F}$ be a set of all functions computed in the circuit's nodes (including inputs). Let $\mathcal{N}$ be a set of all functions computed in the circuit's nonlinear nodes (i.e. any node except XOR, NOT) together with functions returning input bits and the constant* 1 *function. Then the linear subspaces spanned by $\mathcal{F}$ and by $\mathcal{N}$ are the same and, therefore,* $\dim span(\mathcal{F}) = \dim span(\mathcal{N}) \le |\mathcal{N}|$.

Proof. We need to prove that $span(\mathcal{F}) = span(\mathcal{N})$. It is sufficient to show that any function from $\mathcal{F}$ belongs to $span(\mathcal{N})$. We prove by induction on circuit levels. The first level contains input nodes. They belong to $\mathcal{N}$ by definition. Assume that the proposition is proved for k levels and consider $(k+1)$-th level. A XOR node on $(k+1)$-th level corresponds to a linear combination of functions from lower levels which belong to $span(\mathcal{N})$ by induction hypothesis. Therefore, each such XOR node computes a function from $span(\mathcal{N})$ as well. Similarly, any NOT node from $(k+1)$-th level corresponds to a linear combination of a function from k-th level and the constant 1 function. Any other node belongs to $\mathcal{N}$ by definition.

We conclude that circuits for evaluating operations on masked values should have fewer nonlinear gates to avoid algebraic attacks. We use lower bounds on the number of nonlinear gates for random Boolean functions and provide a heuristic estimation of the resilience of random encodings to algebraic attacks.

Proposition 2 (heuristic). *Consider a Boolean circuit with n inputs and n outputs computing a random function. Let $\mathcal{F}$ and $\mathcal{N}$ be set of functions computed in the circuit's nodes as in Proposition 1. Any Boolean function g with n inputs can be expressed as a function g' applied to functions from $\mathcal{F}$. It can be expected that for any g there exists such g' of degree at most 3.*

Proof. The heuristic is based on a counting argument. The number of all Boolean functions of n bits is equal to $exp_2(2^n)$. Recall that $\mathcal{N}$ includes the input variables and the constant 1 function. Boyar *et al.* [7] show that almost all Boolean functions with n inputs require at least $2^{n/2} - n - 1$ AND gates. Using this bound we obtain that $|\mathcal{N}| \geq 2^{n/2}$ (except for negligible fraction of Boolean functions). The number of distinct functions in $span(\mathcal{F})$ is equal to $exp_2(\dim span(\mathcal{F}))$ which is upper bounded by $exp_2(|\mathcal{N}|)$ by Proposition 1. By applying cubic functions to $\mathcal{N}$ we can generate $exp_2(\sigma(|\mathcal{N}|, 3)) \approx \exp_2(2^{3n/2}/6)$ functions. It is reasonable to expect few linear dependencies and then $\exp_2(2^{3n/2}/6)$ functions would cover all $\exp_2(2^n)$ Boolean functions. Under this assumption, any Boolean function applied to the inputs can be computed as a cubic function applied to the circuit nodes.

According to the proposition, the generalized linear algebra attack from Section 3.2 can always be applied to random encodings with linearization degree at most 3. Still, the counting argument shows that random encodings may be useful for protecting against the degree 1 attack and possibly against degree 2 attack.

Goudarzi *et al.* [17] described a heuristic method for constructing circuits for n-bit S-Boxes with approximately $\sqrt{n}2^{n/2+1}$ AND gates. At most $exp_2(2^{n/2+1})$ out of $exp_2(2^n)$ distinct functions may be computed from linear combinations of such circuit's nodes. Therefore, such encoding should be secure against the linear algebraic attack. After a circuit for *Eval* is generated, its security must be evaluated against the attack. For a compositional security we would need to compose each $Eval_\diamond$ with a refresh-gadget and evaluate this composition against the attack as well. Such gadget can be constructed similarly to *Eval* by constructing a look-up table and then synthesizing a circuit for it.

Unless a special method for constructing circuits from look-up tables is developed, we expect that random encodings are useful for at most second-order security. The minimalist quadratic masking described above suits better for first-order protection from performance considerations. However, random encodings may offer additional security: the white-box designer may keep them secret. This will prevent attacks exploiting known structure patterns in the masking circuit. However, it is related to the *structure hiding* problem which we do not consider here. We also expect that it is possible to find random 4-bit encodings with circuits satisfying second-order algebraic security. It may be inefficient to generate random encoding for each node in the reference circuit, but even a single public encoding can be useful as a second-order protection.

FHE-based Encodings. We recall that FHE schemes are a good solution to the full "Value Hiding" problem. The FHE scheme even does not have to be secure in the usual sense: the security parameters may be degraded to have ciphertexts of just a few bits. It is highly likely that after such extreme degradation the scheme will become weak: ciphertext bits will correlate with the secret bit. As we have shown, this problem may be solved by applying a linear masking scheme on top. For the algebraically secure scheme the only requirement is that no predictable value should be computable from the nodes of the modified circuit using low-degree functions.

We describe a simple FHE-style encoding. Choose two small distinct odd primes p, q and let $n = pq$. Define the encoding of a bit $\tilde{v} \in \{0, 1\}$ to be any value v in $\{0, \ldots, n-1\}$ such that $Dec(v) = v \mod p = \tilde{v}$. The expressions for masked operations together with a refreshing gadget are given in Figure 2.

Fig. 2: Modular Masking Scheme.

$$\begin{aligned} Dec(v) &= \tilde{v} \mod p, & (5)\\ Eval_{XOR}(v_1, v_2) &= (v_1 + v_2 - 2v_1v_2) \mod n, & (6)\\ Eval_{AND}(v_1, v_2) &= (v_1v_2) \mod n, & (7)\\ Refresh(v, pr) &= (v + pr) \mod n. & (8) \end{aligned}$$

It is easy to verify that $Eval_{XOR}$ and $Eval_{AND}$ satisfy the requirements from the Definition 1. The randomness used in the encoding can be retrieved as $v \mod q$. As randomness can degrade to 0 or 1 over time, it should be refreshed by adding a fresh random multiple of p to the encoded value.

The security of the scheme depends on the circuit implementing the masked operations using modular additions and multiplications. Each implementation has to be evaluated against the (generalized) linear attack. According to our experiments, setting $n = 3 \times 5$ is enough for protection against the linear attack. This setting results in 4-bit nonlinear encoding. We expect that for larger values of n the scheme has higher order algebraic security. Indeed, the remainder mod p function has full algebraic degree and security of the scheme depends only on how it is implemented. However, the arithmetic incurs a noticeable overhead. For small values of n it may be possible that generic method of synthesizing circuits with few AND gates (e.g. by Goudarzi *et al.* [17]) would be more efficient.

5 Conclusions

In this paper we investigated the possibility of using masking techniques for white-box implementations. We presented several attacks applicable in different scenarios. As a result we obtain requirements for a masking scheme to be useful. We divide the requirements into value hiding and structure hiding. Furthermore, we show that value hiding may be achieved using an algebraically secure nonlinear masking scheme and a classic linear masking scheme. We suggest three constructions for algebraically secure nonlinear masking. Therefore, a value hiding protection can be implemented. We leave the structure hiding problem as a future work.

Acknowledgement

The work of Aleksei Udovenko is supported by the Fonds National de la Recherche, Luxembourg (project reference 9037104).

References

1. O. Billet, H. Gilbert, and C. Ech-Chatbi. *Cryptanalysis of a White Box AES Implementation*, pages 227–240. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

2. A. Biryukov, C. Bouillaguet, and D. Khovratovich. Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key (extended abstract). In P. Sarkar and T. Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 63–84, Kaoshiung, Taiwan, R.O.C., Dec. 7–11, 2014. Springer, Heidelberg, Germany.
3. A. Biryukov, D. Khovratovich, and L. Perrin. Multiset-Algebraic Cryptanalysis of Reduced Kuznyechik, Khazad, and secret SPNs. *IACR Transactions on Symmetric Cryptology*, 2016(2):226–247, 2017.
4. A. Biryukov and A. Shamir. Structural cryptanalysis of SASAS. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 394–405, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.
5. J. W. Bos, C. Hubain, W. Michiels, and P. Teuwen. Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough. In B. Gierlichs and A. Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, volume 9813 of *Lecture Notes in Computer Science*, pages 215–236. Springer, 2016.
6. P. Bottinelli and J. W. Bos. Computational aspects of correlation power analysis. *Journal of Cryptographic Engineering*, 7(3):167–181, Sep 2017.
7. J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of boolean functions over the basis $(\wedge, \oplus, 1)$. 235:43–57, 03 2000.
8. J. Bringer, H. Chabanne, and E. Dottax. White Box Cryptography: Another Attempt. Cryptology ePrint Archive, Report 2006/468, 2006. `http://eprint.iacr.org/2006/468`.
9. B. Carmer, A. J. Malozemoff, and M. Raykova. 5gen-c: Multi-input functional encryption and program obfuscation for arithmetic circuits, 10 2017.
10. S. Chow, P. Eisen, H. Johnson, and P. C. van Oorschot. *A White-Box DES Implementation for DRM Applications*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
11. S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot. White-box cryptography and an AES implementation. In K. Nyberg and H. M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 250–270, St. John's, Newfoundland, Canada, Aug. 15–16, 2003. Springer, Heidelberg, Germany.
12. Y. De Mulder, B. Wyseur, and B. Preneel. *Cryptanalysis of a Perturbated White-Box AES Implementation*, pages 292–310. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
13. ECRYPT-CSA consortium. CHES 2017 Capture The Flag Challenge. The WhibOx Contest, 2017. `http://whibox.cr.yp.to/`.
14. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Oct 2013.
15. H. Gilbert, J. Plût, and J. Treger. *Key-Recovery Attack on the ASASA Cryptosystem with Expanding S-Boxes*, pages 475–490. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.
16. L. Goubin, P. Paillier, M. Rivain, and J. Wang. Reveal Secrets in Adoring Poitras. A victory of reverse engineering and cryptanalysis over challenge 777, 2017. CHES 2017 Rump Session, slides. `https://ches.2017.rump.cr.yp.to/a905c99d1845f2cf373aad564ac7b5e4.pdf`.
17. D. Goudarzi and M. Rivain. *On the Multiplicative Complexity of Boolean Functions and Bitsliced Higher-Order Masking*, pages 457–478. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.
18. Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. *Private Circuits II: Keeping Secrets in Tamperable Circuits*, pages 308–327. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
19. Y. Ishai, A. Sahai, and D. Wagner. *Private Circuits: Securing Hardware against Probing Attacks*, pages 463–481. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
20. M. Karroumi. Protecting white-box AES with dual ciphers. In K. H. Rhee and D. Nyang, editors, *Information Security and Cryptology - ICISC 2010 - 13th International Conference,*

Seoul, Korea, December 1-3, 2010, Revised Selected Papers, volume 6829 of *Lecture Notes in Computer Science*, pages 278–291. Springer, 2010.

21. T. Lepoint and M. Rivain. Another Nail in the Coffin of White-Box AES Implementations. Cryptology ePrint Archive, Report 2013/455, 2013. `http://eprint.iacr.org/2013/455`.
22. H. E. Link and W. D. Neumann. Clarifying obfuscation: improving the security of white-box DES. In *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, volume 1, pages 679–684 Vol. 1, April 2005.
23. B. Minaud, P. Derbez, P.-A. Fouque, and P. Karpman. Key-recovery attacks on ASASA. In T. Iwata and J. H. Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 3–27, Auckland, New Zealand, Nov. 30 – Dec. 3, 2015. Springer, Heidelberg, Germany.
24. Y. Xiao and X. Lai. A Secure Implementation of White-Box AES. In *2009 2nd International Conference on Computer Science and its Applications*, pages 1–6, Dec 2009.