Attacks and Countermeasures for White-box Designs*

Alex Biryukov[1](✉) and Aleksei Udovenko[2](✉)

[1] SnT and CSC, University of Luxembourg, Esch-sur-Alzette, Luxembourg
alex.biryukov@uni.lu
[2] SnT, University of Luxembourg, Esch-sur-Alzette, Luxembourg
aleksei.udovenko@uni.lu

Abstract. In 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 the 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.* [7] proposed a generic attack on white-box primitives called differential computation analysis (DCA). This attack was applied to many white-box implementations both from academia and industry. The attack comes from the area of side-channel analysis and the most common method protecting against such attacks is masking, which in turn is a form of secret sharing. In this paper we 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.

Based on the new constraints, we develop a general method for protecting white-box implementations. We split the protection into two independent components: value hiding and structure hiding. Value hiding must provide protection against passive DCA-style attacks that rely on analysis of computation traces. Structure hiding must provide protection against circuit analysis attacks. In this paper we focus on developing the value hiding component. It includes protection against the DCA attack by Bos *et al.* and protection against a new attack called algebraic attack.

We present a provably secure first-order protection against the new algebraic attack. The protection is based on small gadgets implementing secure masked XOR and AND operations. Furthermore, we give a proof of compositional security allowing to freely combine secure gadgets. We derive concrete security bounds for circuits built using our construction.

Keywords: White-box, Obfuscation, Cryptanalysis, Provable Security, Masking

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

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.* [14,15] 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.* [10]. Instead of look-up tables, the cipher is represented as a sequence of functions over $\mathbb{F}_{2^n}$ 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 [2,16,28]. Consequent attempts to fix them were not successful [27,33]. Moreover, Michiels *et al.* generalized the attack by Billet *et al.* [2] and showed that the approach of Chow *et al.* is not secure for any SPN cipher with MDS matrices. This follows from the efficient cryptanalysis of any SASAS structure [5]. Recently several white-box schemes based on the ASASA structure were proposed [3]. However the strong white-box scheme from that paper was broken [4,21,29] (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. [20]).

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.* [7] 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). Sasdrich *et al.* [30] pointed out that the weakness against the DCA attack can be explained using the Walsh transform of the encoding functions. Banik *et al.* [1] analyzed software countermeasures against the DCA attack and proposed another automated attack called Zero Difference Enumeration attack.

In light of such powerful automated attack the question arises: how to create a whitebox scheme secure against the DCA attack? The most common countermeasure against side-channel attacks is masking, which is a form of secret sharing. It is therefore natural 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 [13, 20].

In this paper we investigate the possibility of using masking schemes in the white-box setting. We restrict our analysis to implementations in the form of Boolean circuits. Our contribution splits into three parts:

1. **Attacks on Masked White-Box Implementations.** In Section 3 we develop a more generic DCA framework and describe multiple generic attacks against masked implementations. The attacks show that the classic Boolean masking (XOR-sharing) is inherently weak. Previous and new attacks are summarized in Table 1. We remark that conditions for different attacks vary significantly and the attacks should not be compared solely by time complexity. For example, the fault-based attacks are quite powerful, but it is relatively easy to protect an implementation from these attacks. From our attacks we conclude that more general nonlinear encodings are needed and we deduce constraints that a secure implementation must satisfy. We believe that our results provide new insights on the design of white-box implementations. We note that a basic variant of the (generalized) linear algebra attack was independently discovered by Goubin *et al.* [23].
2. **Components of Protection.** We propose in Section 4 a general method for designing a secure white-box implementation. The idea is to split the protection into two independent components: *value hiding* and *structure hiding.* The value hiding component must provide protection against passive DCA-style attacks - attacks that rely solely on analysis of computed values. In particular, it must provide security against the correlation attack and the algebraic attack. We suggest that security against these two attacks can be achieved by applying a classic linear masking scheme on top of a nonlinear masking scheme protecting against the algebraic attack. The structure hiding component must secure the implementation against circuit analysis attacks. The component must protect against circuit minimization, pattern recognition, pseudorandomness removal, fault injections, etc. Possibly this component may be splitted into more sub-components (e.g. an integrity protection). Development of a structure hiding protection is left as a future work.
3. **Provably Secure Construction.** Classic t-th order masking schemes protect against adversaries that are allowed to probe t intermediate values computed by the implementation. The complexity of the attack grows fast when t increases. In the new algebraic attack the adversary is allowed to probe all intermediate values but she can combine them only with a function of low algebraic degree d. Similarly, the attack complexity grows fast when d increases and also when the circuit size increases. We develop a framework for securing an implementation against the algebraic attack. We describe a formal security model and prove composability of first-order secure circuits. Finally, we propose a first-order secure masking scheme implementing XOR

and AND operations. As a result, our framework provides provable security against the first-order algebraic attack. We derive concrete security bounds for our construction. Finally, we implement the AES-128 block cipher protected using our new masking scheme.

Table 1: Attacks on masked white-box implementations.

Attack	Ref.	Data	Time
Correlation	[7],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	[23],Sec. 3.2	$\mathcal{O}(n)$	$\mathcal{O}(n^\omega + n^2 k)$
Generalized Lin. Alg.	[23],Sec. 3.2	$\mathcal{O}(\sigma(n,d)))$	$\mathcal{O}(\sigma(n,d)^\omega + \sigma(n,d)^2 k)$
LPN-based Gen. Lin. Alg.	Sec. 3.2	$D_{LPN}(r, \sigma(n,d))$	$T_{LPN}(r, \sigma(n,d))$
1-Share Fault Injection	Sec. 3.3	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
2-Share Fault Injection	Sec. 3.3	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$

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; r is the noise ratio in the system of equations, $T_{LPN}(r,m), D_{LPN}(r,m)$ are time and data complexities of solving an LPN instance with noise ratio r and m variables.

A code implementing the described attacks, verification of the algebraic masking schemes and the masked AES-128 implementation is publicly available at [6]:

`https://github.com/cryptolu/whitebox`

Outline We provide the notations in Section 2. The general attack setting and attacks are described in Section 3. We discuss a general method for securing a white-box design in Section 4. In Section 5 we develop countermeasures against the algebraic attack. Finally, we conclude and suggest future work in Section 6.

2 Notations and Definitions

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

- $\wedge, \vee, \oplus$ denote Boolean AND, OR and XOR respectively.

- $\mathbb{F}_2$ 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 $v = (v_1, \ldots, v_n)$.
- $|X|$ denotes the size of the vector/set X. If X is a circuit, $|X|$ denotes the number of nodes in it.
- The *Weight* of a vector v is the number or nonzero entries in it and is denoted $wt(v)$. Weight of a Boolean function is the weight of its truth table.
- The *Bias* of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is denoted $\mathcal{E}(f)$ and is equal to $|1/2 - wt(f)/2^n|$.
- The *Correlation* of two n-bit vectors v_1 and v_2 is defined as

  $$cor(v_1, v_2) = \frac{n_{11}n_{00} - n_{01}n_{10}}{\sqrt{(n_{00} + n_{01})(n_{00} + n_{10})(n_{11} + n_{01})(n_{11} + n_{10})}},$$

  where n_{ij} denotes the number of positions where v_1 equals to i and v_2 equals to j. If the denominator is zero then the correlation is set to zero. *cor* is the sample Pearson correlation coefficient of two binary variables, also known as the Phi coefficient. Other correlation coefficients may be used, see e.g. [32].
- $\mathbf{0}, \mathbf{1}$ denote the two constant functions.
- 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 the *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 f 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. It is denoted $\deg f$.
- $\sigma(n, d) = \sum_{i=0}^{d} \binom{n}{i}$ is the number of monomials of n bit variables from $\mathbb{F}_2$ of degree at most d.
- Let $\mathcal{V}$ be a set of Boolean functions with the same domain $\mathbb{F}_2^n$. Define the *d-th order closure of* $\mathcal{V}$ (denoted $\mathcal{V}^{(d)}$) to be the vector space of all functions obtained by composing any function of degree at most d with functions from $\mathcal{V}$:

  $$\mathcal{V}^{(d)} = \{f \circ (g_1, \ldots, g_{|\mathcal{V}|}) \ \mid \ \forall f : \mathbb{F}_2^{|\mathcal{V}|} \to \mathbb{F}_2, \deg f \leq d, g_i \in \mathcal{V}\},$$

  where $g_i : \mathbb{F}_2^n \to \mathbb{F}_2$ and $h = f \circ (g_1, \ldots, g_{|\mathcal{V}|})$, $h : \mathbb{F}_2^n \to \mathbb{F}_2$ is such that $h(x) = f(g_1(x), \ldots, g_{|\mathcal{V}|}(x))$. For example,
  - $\mathcal{V}^{(1)}$ is spanned by $\{\mathbf{1}\} \cup \{g_i \ \mid \ g_i \in \mathcal{V}\}$,
  - $\mathcal{V}^{(2)}$ is spanned by $\{\mathbf{1}\} \cup \{g_i g_j \ \mid \ g_i, g_j \in \mathcal{V}\}$, etc. (includes $\mathcal{V}^{(1)}$ as $i = j$ is allowed).

3 Attacks On Masked Implementations

We describe the general setting for our attacks. We consider *a keyed symmetric primitive*, e.g. a block cipher. A *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.*

Definition 1. *A Boolean circuit C is a directed acyclic graph where each node with the indegree $k > 0$ has an associated k-ary symmetric Boolean function. Nodes with the indegree equal to zero are called inputs of C and nodes with the outdegree equal to zero are called outputs of C.*

Let $x = (x_1, \ldots, x_n)$ (resp. $y = (y_1, \ldots, y_m)$) be a vector of input (resp. output) nodes in some fixed order. For each node v in C we say that it computes a Boolean function $f_v : \mathbb{F}_2^n \to \mathbb{F}_2$ defined as follows:

- *for all $1 \leq i \leq n$ set $f_{x_i}(z) = z_i$,*
- *for all non-input nodes v in C set $f_v(z) = g(f_{c_1}(z), \ldots, f_{c_k}(z))$, where $c_1, \ldots, c_k$ are nodes having an outgoing edge to v.*

The set of f_v for all nodes v in C is denoted $\mathcal{F}(C)$ and the set of f_{x_i} for all input nodes x_i is denoted $\mathcal{X}(C)$. By an abuse of notation we also define the function $C : \mathbb{F}_2^n \to \mathbb{F}_2^m$ as $C = (f_{y_1}, \ldots, f_{y_m})$.

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. Locating and manipulating the pseudorandomness generation is a possible attack direction. However, as we aim to study the applicability of masking schemes, 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 (for example, at the same circuit level or at a short distance in the circuit graph). 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 [9]). 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 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 the case of the single window coverage.

General DCA attack. We would like to note that the term "differential computation analysis" (DCA) is very general. In [7] 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 adapt 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 Algorithm 1.

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 the 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 the 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 vectors $\tilde{V}$.
```

We remark that the correlation-based DCA attack from [7] can be implemented on-the-fly, without computing the full vectors v_j. In contrast, most of our attacks require 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.

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 [7] 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 [7]. 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 t-bit 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 or the maximum value of its absolute entries.

We assume that the predictable value is not highly unbalanced. Then for the attack to succeed we need the correlated shares to hit at least one combination m a 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 [7] relied on single-bit correlations and still was successfully applied to break multiple 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 the 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. For simplicity, we assume the data complexity is $O(1)$ and 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 the 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 one of predictable vectors. Clearly, this is a basic linear algebra problem. Let M be the matrix that has as columns vectors from the current window. For each predictable vector $\tilde{v}$ we solve the equation $M \times x = \tilde{v}$. A 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 in [23] and among other techniques was successfully applied [22] during the WhibOx 2017 competition [17] in order to break the winning challenge "Adoring Poitras".

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 remark that it is enough to consider only nonlinear (e.g. AND, OR) and input nodes inside the current window. All other nodes are affine combinations of these and are redundant. We formalize this fact in the following proposition.

Proposition 1. *Let C be a Boolean circuit. Let $\mathcal{N}(C)$ be the set of all functions computed in the circuit's nonlinear nodes (i.e. any node except XOR, NOT,* *NXOR) together with functions returning input bits. Then for any integer $d \geq 1$ the sets $\mathcal{F}^{(d)}(C)$ and $\mathcal{N}^{(d)}(C)$ are the equal.*

Proof. Note that for any set $\mathcal{V}$ we have $\mathcal{V}^{(d)} = (\mathcal{V}^{(1)})^{(d)}$. Therefore, we only need to prove that $\mathcal{F}^{(1)}(C) = \mathcal{N}^{(1)}(C)$. It is sufficient to show that any function from $\mathcal{F}$ belongs to $\mathcal{N}^{(1)}(C)$. This can be easily proved by induction on circuit levels. □

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 nonlinear 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 2^8 products from each group and no products across the 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.

Value-restriction analysis. The described algebraic attack can be modified to cover a broader range of masking schemes. Consider a low-degree combination of vectors from the current window and assume that the function it computes can be expressed as $s \wedge r$, where s is the correct predictable value and r is some uniform pseudorandom (unrelated) value. The basic algebraic attack will not succeed because $s \wedge r$ is not always equal to the predictable value s. However, it is possible to extend the attack to exploit the leakage of $s \wedge r$. The adversary chooses a set of inputs for which the predictable value s is equal to 0 and adds a single random input for which the predictable value is equal to 1 (the adversary may need to guess a part of the key to compute the predictable value). Then with probability 1/2 he is expected to find a vector with all bits equal to 0 except the last bit equal to 1. In case the predictable value is wrong, the chance of finding such vector is exponentially small in the size of the plaintext set. The same approach works for more complex leaked functions. In particular, the leaked function may depend on multiple predictable values, e.g. on all output bits of an S-Box. The only requirement is that the leaked function must be constant for at least one assignment of the predictable values (except of course the case when the leaked function is constant on all inputs). Note that the adversary must be able to find the correct assignment of predictable values. As a conclusion, this attack variant reveals a stronger constraint that a masking scheme must satisfy in order to be secure.

Algebraic attack in the presence of noise. In spirit of the value-restriction analysis, we continue to explore classes of exploitable leaking functions. Assume that a low-degree combination of vectors from the current window corresponds to a function $s \oplus e$, where s is the correct predictable vector and e is a function with a low Hamming weight. The function e may be unpredictable and we consider it as noise. The problem of solving a noisy system of linear equations is well known as Learning Parity with Noise (LPN). It is equivalent to the problem of decoding random linear codes. The best known algorithms have exponential running time. We refer to a recent result by Both *et al.* [8] where the authors propose an algorithm with approximated complexity $2^{1.3nr}$, where n is the number of unknown variables and r is the noise ratio. Several algorithms with low memory consumption were recently proposed by Esser *et al.* [18]. The best algorithm for the problem depends on the exact instance parameters. The number of variables in our case corresponds to the number of monomials considered, i.e. the window size n in the linear attack and $\sigma(n, d)$ in the generalized attack. For example, if a linear combination of vectors from a 100-node window leaks s with noise ratio 1/4 then the LPN-based attack will take time $2^{32.5}$ using the algorithm from [8].

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 the 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. We assume 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 in the output any information about the fault. 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 adapted 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 other 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 may need to be faulted in order to succeed. The goal is to get the same faulted output by flipping different nodes or wires as such an event 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 should be high enough to avoid combinatorial attacks. Moreover, the minimum number of shares that correlate with the reference circuit values should be high as well.
2. There should be no low-degree decoders in order to prevent the algebraic attack.
3. The circuit must not admit analysis that 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 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 [20] 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 suggest a modular approach to solve this problem. We split the problem into two components:

1. *(Value Hiding)* Protection against generic passive attacks that 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.* [26]). 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 (even though the adversary may learn the key from the decryption phase, the locations of intermediate shares are unknown and the scheme may remain secure). However, this direction leads to very inefficient schemes: typical FHE schemes have very large ciphertexts and complex circuits. This contradicts our goal to investigate schemes with reasonable number of shares.

We suggest to further split the first component into two parts. The first part is protection against 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. Though one has to be careful and avoid very strong correlation, otherwise the LPN-based variant of the algebraic attack may be applicable. 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 [25]. 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.

We note that it is not possible to formally separate value hiding from structure hiding. If we give the adversary computed vectors of values even in shuffled order, she can reconstruct the circuit in reasonable time and then analyze it. One possible direction is to mix the value vectors linearly by a random linear mapping before giving to the adversary. It may be a difficult problem for the adversary to recover the circuit or its parts from such input. However, such model makes the correlation DCA attack almost inapplicable, since a lot of values are unnaturally mixed up and the correlations are not predictable, even though it is perfectly possible that the original unmixed values have strong correlations with secret variables.

5 Algebraically Secure Masking Schemes

The algebraic attack is very powerful and the classic XOR-sharing masking schemes can not withstand it. Therefore, it is important to develop new masking schemes which are secure against the algebraic attack. In this section we formalize security against the algebraic attack and propose a provably first-order secure construction.

We start by describing the attack model and formalizing security against the algebraic attack in Section 5.1. Ways of proving security in the new model are developed in Section 5.2. Next, we analyze composability in Section 5.3. An algorithm for checking security of gadgets is proposed in Section 5.4. Finally, we propose a concrete secure gadget in Section 5.5.

5.1 Security Model

We extract a subproblem from the whitebox design problem. Recall that during the algebraic attack, the adversary tries to find a function f of low degree d such that when applied to values computed in the nodes of the obfuscated circuit it would produce some predictable value. Typically, predictable value is a value computed using the *reference circuit* and it depends on a small fraction of the key. Our aim is to "hide" predictable values among unpredictable values. The unpredictability of computed functions may only come from the secret key/randomness used during the obfuscation process. In order to develop a formal attack model we allow the obfuscated circuit to use *random* bits. We underline that randomness here is merely an abstraction required for provable security arguments.

In the real whitebox implementation the random bits may be implemented as *pseudorandom* values computed from the input. Of course the pseudorandom generation part has to be protected as well. However, the white-box designer is free to choose arbitrary pseudorandom generator and its protection is an easier task then obfuscating a general circuit. For example, the designer can choose a random circuit satisfying some basic properties like computing a balanced function. The resulting circuit protected against the algebraic attack using pseudorandomly generated bits must further be obfuscated and protected from *removal* of the pseudorandomness. This is type of protection that we called *structure hiding* in Section 4 and it is out of scope of this paper.

We note a strong similarity between the algebraic attack and the side channel probing attack. In the t-th order probing attack the adversary may observe t intermediate values computed in the circuit. In the d-th order algebraic attack the adversary may observe all intermediate values but she can combine them only with a function of degree at most d.

The main idea of masking schemes is to *hide the values* computed in the reference circuit using (pseudo)random masks. We assume that the adversary knows the reference circuit. Given the inputs (e.g. a plaintext and a key) she can compute all intermediate values. The final goal of the adversary is to recover the key of an obfuscated implementation or, at least, learn some partial information about it. To formalize this, we adapt classic semantic security and indistinguishability ideas. The adversary may ask to encrypt two different vectors of inputs. The challenger chooses randomly one of the vectors and provides an oracle modelling the algebraic attack to the adversary. The goal of the adversary is to decide which of the vectors was encrypted. If she can not do this, then she can not learn any information about the hidden inputs (e.g. the plaintext and the key). Note that in this model the adversary may choose many different keys which is not possible in the white-box scenario. However, it leads to simpler definitions since we do not have to distinguish plaintext and key and we just treat them as one input. It is possible to add a constraint allowing to choose only a single key per input vector, but this would not lead to any improvement.

The oracle modelling the algebraic attack should not reveal too much information about computed values. Otherwise, it may be possible for the adversary to reconstruct the obfuscated circuit and then we would arrive in the general white-box scenario. We model the attack as follows: the adversary chooses the target function among linear (or higher-order) combinations of the intermediate functions in the circuit and she tries to guess its values during encryptions of the inputs from one of the two vectors. Note that some functions may have strong correlation with some function of the input. For a small vector of inputs the adversary may simply guess the value, ask the oracle a few times until the guess is correct and then compute the correlations. However, in the real algebraic attack this is not possible due to presence of ”noise” in the circuit. For a small number of plaintexts there will be a lot of false matches for any ”predicted” value, because there are many different functions computed in the circuit and it is highly probable that there is a linear combination of them matching an arbitrary value. We take this into account and require that only the function chosen by the adversary has to match the predicted value. As a result, the adversary can not accurately predict values of any *single* function in the d-th order closure of the circuit functions in order to run the linear algebra attack.

The circuit in the model can not take the input as it is, because these values allow for a simple distinguisher. Since we are developing a masking scheme, we assume that the inputs are already masked using random shares. This goes in parallel with the classic Boolean masking scenarios. We would like to stress that this is necessary in order to formally analyze the security of masked computations. Therefore, we do not consider the initial masking and the final un-masking processes. Indeed, these procedures are not relevant for the algebraic attack since they are not related to the reference circuit.

Taking into account the above discussions, we propose the following game-based security definition:

Definition 2 (Prediction Security (d-PS)). *Let* $C : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^{M}$ *be a Boolean circuit,* $E : \mathbb{F}_2^{N} \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ *an arbitrary function,* $d \geq 1$ *an integer and* $\mathcal{A}$ *an adversary. Consider the following security game:*

```
Experiment $\mathsf{PS}^{C,E,d}(\mathcal{A}, b)$:
    $(\tilde{f}, x^{[0]}, x^{[1]}, \tilde{y}) \leftarrow \mathcal{A}(C, E, d)$, where
        $\tilde{f} \in \mathcal{F}^{(d)}(C), x^{[l]} = (x_1^{[l]}, \ldots, x_Q^{[l]}), x_i^{[l]} \in \mathbb{F}_2^N, \tilde{y} \in \mathbb{F}_2^Q$
    $(r_1, \ldots, r_Q) \xleftarrow{\$} (\mathbb{F}_2^{R_E})^Q$
    $(\tilde{r}_1, \ldots, \tilde{r}_Q) \xleftarrow{\$} (\mathbb{F}_2^{R_C})^Q$
    For any $f \in \mathcal{F}^{(d)}(C)$ define
        $y(f) = \Big(f\big(E(x_1^{[b]}, r_1), \tilde{r}_1\big), \ldots, f\big(E(x_Q^{[b]}, r_Q), \tilde{r}_Q\big)\Big)$
    $F \leftarrow \{f \in \mathcal{F}^{(d)}(C) \mid y(f) = \tilde{y}\}$
    return $F = \{\tilde{f}\}$
```

In the above experiment, $\xleftarrow{\$}$ *means sampling uniformly at random. Define the advantage of an adversary* $\mathcal{A}$ *as*

$$\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] = \left|\mathsf{Pr}[\mathsf{PS}^{C,E,d}(\mathcal{A},0) = 1] - \mathsf{Pr}[\mathsf{PS}^{C,E,d}(\mathcal{A},1) = 1]\right|.$$

The pair (C, E) *is said to be d-th order prediction-secure (d-PS) if for any adversary* $\mathcal{A}$ *the advantage is negligible.*

Example. Consider a white-box AES implementation with a first-order Boolean masking protection. Assume that there are two nodes in the circuit computing two masks of an output bit of an S-Box in the first round. Denote the functions computed by masks as f_1, f_2. The adversary finds these nodes and chooses $\tilde{f} = f_1 \oplus f_2 \in \mathcal{F}^{(1)}(C)$. She also chooses sufficiently large Q and random vectors $x^{[0]}$ and $x^{[1]}$ of Q (plaintext, key) pairs. For example, the same key may be used for all pairs in $x^{[0]}$ and another key for all pairs in $x^{[1]}$. The adversary computes $\tilde{y} = (s(x^{[0]}_1), ..., s(x^{[0]}_Q))$ (where function s computes the output bit of the attacked S-Box in the first round from the plaintext and the key). In this case the adversary succeeds in the game with advantage close to 1 and the implementation is not prediction-secure (indeed, the adversary easily distinguishes the two keys). Note that we required the adversary to find the nodes in order to choose the right function $\tilde{f}$. Since the adversary is unbounded, this is just a technical requirement. In the real attack the adversary does not need to guess the function.

The function E in the definition should be referred to as *an encoding function.* Though the definition allows the encoding function to be arbitrary, we are mainly interested in the encodings with useful semantics, i.e. masking. Moreover, we expect the encoding to be lightweight and universal: main computations should be performed in the circuit C.

The circuit C can be completely unobfuscated but still prediction-secure, because the adversary is forced to consider the whole vector space $\mathcal{F}^{(d)}(C)$. In a real white-box implementation this restriction is expected to be enforced by the structure-hiding protection.

We now discuss possible attacks that are not covered by this definition. The definition ensures that any single function from $\mathcal{F}^{(d)}(C)$ is unpredictable. However, it may be possible that multiple functions jointly exhibit a behaviour that leads to an attack. For example, the dimension of $\mathcal{F}^{(d)}(C)$ may differ depending on the input being encoded. The definition also does not cover the LPN-based attack.

5.2 Security Analysis

In the experiment both the encoding function E and the circuit C use randomness. However, the d-th order closure is computed only using functions from $\mathcal{F}(C)$. Still, the inputs of C include the outputs of E and that is how the randomness used in E affects the computations in C. In other words, E generates some distribution in the inputs of C. Therefore, in order to study functions from $\mathcal{F}^{(d)}(C)$ we need to compose them with E.

It is crucial to study how functions from $\mathcal{F}^{(d)}(C)$ composed with E behave with a fixed input x. Consider a function $f \in \mathcal{F}^{(d)}(C)$. If the function $f(E(x,\cdot),\cdot)$ is constant for some x and the function $f(E(x',\cdot),\cdot)$ is non-constant for some $x' \neq x$ (or is constant but $f(E(x,\cdot),\cdot) \neq f(E(x',\cdot),\cdot)$, then these inputs are distinguishable and the pair (C,E) is insecure[3]. More generally, if for some $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0},\mathbf{1}\}$ and for some $x \in \mathbb{F}_2^N$ the function $f(E(x,\cdot),\cdot)$ is non-constant but has a high bias (i.e. it has very low or very high weight), then the adversary still may have high chances to predict its output. We conclude that for all functions $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0},\mathbf{1}\}$ and for all $x \in \mathbb{F}_2^N$ the function $f(E(x,\cdot),\cdot)$ should have a low bias.

We now show that this requirement is enough to achieve d-th order prediction security if there are enough random bits used in the main circuit. The following proposition gives an upper bound on d-PS advantage from the maximum bias and the number of random bits.

Definition 3. *Let C, E be defined as above. For any function $f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0},\mathbf{1}\}$ and for any $x \in \mathbb{F}_2^N$ define $f_x : \mathbb{F}_2^{R_E} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2$ given by $f_x(r_e, r_c) = f(E(x, r_e), r_c)$ and denote the set of all such functions $\mathcal{R}$:*

$$\mathcal{R} = \{f(E(x,\cdot),\cdot) \ \mid \ f \in \mathcal{F}^{(d)}(C) \setminus \{\mathbf{0},\mathbf{1}\}, x \in \mathbb{F}_2^N\}.$$

Proposition 2. *Let ε be the maximum bias among all functions from $\mathcal{R}$:*

$$\varepsilon = \max_{f_x \in \mathcal{R}} \mathcal{E}(f_x).$$

Let $e = -\log_2(1/2 + \varepsilon)$. Then for any adversary $\mathcal{A}$ choosing vectors of size Q

$$\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \leq min(2^{Q-R_C}, 2^{-eQ}). \tag{1}$$

Proof. First, we prove that $\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \leq 2^{Q-R_C}$. If $\tilde{f}$ chosen by the adversary is an affine function of random bits r (independent of x), then it is clear that the advantage is zero. Otherwise, we compute the probability of the event when the predicted value $\tilde{y}$ matches some linear function of random bits r. There are R_C independent uniformly distributed random vectors $r_1, \ldots, r_{R_C}$ from $\mathbb{F}_2^Q$. Let p be the probability of the event that they span the whole space $\mathbb{F}_2^Q$. In this case the experiment returns 0, because any $\tilde{y}$ matches a function different from the one chosen by the adversary. The following holds (see e.g. [19]):

$$p = \prod_{i=0}^{Q-1} \left(1 - 2^{i-R_C}\right), \quad \log_2(1-p) \leq Q - R_C.$$

We conclude that $p \geq 1 - 2^{Q-R_C}$ and the advantage is upper bounded by 2^{Q-R_C}.

[3] Unless $f(E(x',\cdot),\cdot)$ has extremely high bias and is indistinguishable from the constant function on practice.

Now we prove that $\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \leq 2^{-eQ}$. We simply bound the probability that the adversary submits $\tilde{f}, \tilde{y}$ such that $y(\tilde{f}) = \tilde{y}$ in the experiment. Since elements of $y(\tilde{f})$ are independent, the probability to have $y(\tilde{f}) = \tilde{y}$ is maximized when each bit of $\tilde{y}$ equals to the most probable value of the respective bit of $y(\tilde{f})$ (the adversary would also need to use the least probable value at least once to avoid matching with the constant functions). For each bit the probability is bounded by $1/2 + \varepsilon$, therefore for Q bits the bound is $(1/2 + \varepsilon)^Q = 2^{-eQ}$. □

Note that the bounds are quite loose. The randomness-based term takes into account only single random bits from r_c. The randomness in the inputs of C (generated from r_e in the encoding process) as well as all intermediate values computed in the circuit add much more noise (note that we can not directly include r_e since it is used in the encoding process and not in the main circuit). The bias-based term bounds only the probability of predicting the output for a single vector of inputs. It does not include the cost of distinguishing the two vectors. We stick to these bounds as our current goal is to provide a simple and sound provably secure protection.

Assume that we know the maximum bias ε in $\mathcal{R}$ and we want to achieve a better security bound. We can always add "dummy" random bits to the circuit. Note that this leads to stronger requirements for the structure-hiding protection. It follows that given the maximum bias, we can compute how many "dummy" random bits are needed to achieve any required security level:

Corollary 1. *Let k be a positive integer. Then for any adversary $\mathcal{A}$*

$$\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \leq 2^{-k} \textit{ if}$$

$$e > 0 \textit{ and } R_C \geq k \cdot (1 + \frac{1}{e}).$$

Proof. Consider each term of the bound from Proposition 2:

$$Q - R_C \leq -k \text{ or } -eQ \leq -k.$$

The result follows from the second term if $Q \geq \frac{k}{e}$. To cover all other Q we need $R_C \geq Q + k \geq k \cdot (1 + \frac{1}{e})$. □

We remark that the advantage bound is information-theoretic as we do not constraint the adversary's powers. This is an effect of the attack formalization given in Definition 2: the attack requires that the adversary predicts the chosen function precisely. An unbounded adversary could simply iterate over all functions $f \in \mathcal{F}^{(d)}(C)$ and e.g. compute the bias. We argue that this kind of attack is not the linear algebra attack that we consider. Furthermore, the attack model restricts the adversary to use the full circuit C. Without this restriction it would be possible to choose a part of the circuit (a *window*) to reduce the noise. In our model we expect that a structure-hiding protection is used to prevent this.

5.3 First-order Secure Construction

Given the notion of prediction security we are now interested in developing secure constructions. A common strategy is to develop small secure circuits (called *gadgets*) and compose them in a provably secure way. Our definition does not immediately lead to composability, because it includes the encoding step which is not expected to be present in the intermediate gadgets. In order to proceed, we split up the prediction security into circuit security and encoding security. The new notions are stronger in order to get proofs of secure composability. Note that they are limited to the first-order security ($d = 1$) and it is not obvious how to extend them to higher orders.

Definition 4 (Circuit Algebraic Security (ε-1-AS)). *Let $C(x, r) : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit. Then C is called first-order algebraically ε-secure (ε-1-AS) if for any $f \in \mathcal{F}^{(1)}(C) \setminus \{\mathbf{0}, \mathbf{1}\}$ one of the following conditions holds:*

1. *f is an affine function of x,*
2. *for any $x \in \mathbb{F}_2^{N'}$ $\mathcal{E}(f(x, \cdot)) \leq \varepsilon$, where $f(x, \cdot) : \mathbb{F}_2^{R_C} \to \mathbb{F}_2$*

Definition 5 (Encoding Algebraic Security (ε-1-AS)). *Let $E(x, r) : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be an arbitrary encoding function. Let $\mathcal{Y}$ be the set of the coordinate functions of E (i.e. functions given by the outputs bits of E). The function E is called a first-order algebraically ε-secure encoding (ε-1-AS) if for any function $f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0}, \mathbf{1}\}$ and any $x \in \mathbb{F}_2^N$ the bias of the function $f(x, \cdot) : \mathbb{F}_2^{R_E} \to \mathbb{F}_2$ is not greater than ε:*

$$\max_{f \in \mathcal{Y}^{(1)} \setminus \{\mathbf{0},\mathbf{1}\}, x \in \mathbb{F}_2^N} \mathcal{E}(f(x, \cdot)) \leq \varepsilon.$$

The following proposition shows that if both an encoding and a circuit are algebraically secure, then their combination is prediction-secure:

Proposition 3. *Let $C : \mathbb{F}_2^{N'} \times \mathbb{F}_2^{R_C} \to \mathbb{F}_2^M$ be a Boolean circuit and let $E : \mathbb{F}_2^N \times \mathbb{F}_2^{R_E} \to \mathbb{F}_2^{N'}$ be an arbitrary encoding function.*

If C is ε_C-1-AS circuit and E is ε_E-1-AS encoding, then

$$\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \leq min(2^{Q-R_C}, 2^{-eQ}),$$

where $e = -\log_2 (1/2 + max(\varepsilon_C, \varepsilon_E))$.

Proof. If the function $\tilde{f}$ chosen by the adversary is an affine combination of the input x of C, then the encoding security of E applies leading to the bound with $e = -\log_2 (1/2 + \varepsilon_E)$. Otherwise, ε_C-1-AS security of C provides the bound with $e = -\log_2 (1/2 + \varepsilon_C)$ (the bias bound applies for any fixed input x of C, therefore it applies for any distribution of x generated by E as well). □

Finally, we show that ε-1-AS circuits are composable, i.e. are secure gadgets. We can compose gadgets in arbitrary ways and then join the final circuit with a secure encoding function to obtain a prediction-secure construction.

Proposition 4 (**ε-1-AS Composability**). *Consider ε-1-AS circuits $C_1(x_1, r_1)$ and $C_2(x_2, r_2)$. Let C be the circuit obtained by connecting the output of C_1 to the input x_2 of C_2 and letting the input r_2 of C_2 be the extra input of C:*

$$C(x_1, (r_1, r_2)) = C_2(C_1(x_1, r_1), r_2).$$

Then $C(x_1, (r_1, r_2))$ is also a ε-1-AS circuit.

Proof. Consider an arbitrary function $\tilde{f}(x_1, r_1, r_2) \in \mathcal{F}^{(1)}(C)$. By linearity, it can be written as $u \oplus v$, where $u \in \mathcal{F}^{(1)}(C_1)$ and v is a function from $\mathcal{F}^{(1)}(C_2)$ composed with C_1 (by connecting the output of C_1 to the input x_2 of C_2). Since C_2 is ε-1-AS, v is either an affine function of x_2 (which belongs to $\mathcal{F}^{(1)}(C_1)$) or has a bias not greater than ε when x_2 is fixed (i.e. when x_1, r_1 are fixed). In the first case, we get that $\tilde{f}$ belongs to $\mathcal{F}^{(1)}(C_1)$ and security follows from ε-1-AS security of C_1. In the second case, observe that the bias of v can not exceed ε for any fixed x_2 and, therefore, it can not exceed ε for any distribution of x_2. Moreover, u is independent from r_2. Therefore, the bias of $\tilde{f} = u \oplus v$ is not greater than the bias of v which is bounded by ε since C_2 is a ε-1-AS circuit. □

This result shows that due to frequent use of fresh randomness it is guaranteed that the maximum bias does not grow when we build large algebraically secure circuits from smaller ones. It means that ε-1-AS circuits offer a solid protection against the LPN-based variant of the algebraic attack as well. The complexity of LPN algorithms grows exponentially with the number of unknowns. Therefore, increasing the number of random nodes as suggested by the Corollary 1 allows to reach any required level of security against LPN attacks at the same time. Exact required number of random nodes depends on the maximum bias ε and chosen LPN algorithm.

5.4 Verifying Algebraic Security

Proposition 4 shows that we can compose algebraically secure circuits. Large circuits can be constructed from a set of *gadgets* - small algebraically secure circuits with some useful semantics. In order to design new gadgets we need to be able to check their algebraic security. The simplest way to get a bound on bias is based on looking at the algebraic degree of computed functions: the minimum weight of a nonzero function of n bits of degree d is equal to 2^{n-d} (see e.g. [12]). Therefore, we can think about the following algorithm for checking a circuit $C(x, r_C)$: for any fixed input x compute the ANFs of the functions computed in $C(x, \cdot)$ (functions of r_C) and return the maximum observed degree. The degree can not grow when functions are combined linearly. Therefore, the bias bound can not grow as well, except when the resulting function is constant in which case the bias is maximal and the gadget may be insecure. As a result, our method for verifying algebraic security splits into two parts:

1. verify that there is no bias equal to 1/2 among restrictions of functions from $\mathcal{F}^{(1)}(C)$ except the constant functions and affine functions of x;

2. compute the maximum degree among all restrictions of the intermediate functions and compute the corresponding bias bound.

The second step is straight-forward. We describe an algorithm that solves the first step.

Consider a circuit $C(x, r) : \mathbb{F}_2^N \times \mathbb{F}_2^R \to \mathbb{F}_2^M$. For all $c \in \mathbb{F}_2^N$ let L_c be the linear map that returns the restriction $x = c$ of a function f from $\mathcal{F}^{(1)}(C)$ (e.g. if functions are represented as truth table vectors then L_c returns the truth table entries corresponding to the case $x = c$). Note that the domain of L_c is defined to be the subspace $\mathcal{F}^{(1)}(C)$.

We now give an equivalent condition for the first part of the verification. It serves as a basis for the verification algorithm given in Algorithm 2.

Proposition 5. *The circuit C is ε-1-AS for some $\varepsilon < 1/2$ if and only if for all c the following holds:*

$$\dim \ker L_c = N. \tag{2}$$

Proof. For any $c \in \mathbb{F}_2^N$ let F_c be the subspace of $\mathcal{F}^{(1)}(C)$ containing functions that are constant when x is fixed to c. Also let $F = \bigcup_c F_c$. $\varepsilon < 1/2$ requires that any $f \in \mathcal{F}^{(1)}(C)$ either belongs to $\mathcal{X}^{(1)}(C)$ or is non-constant for any fixed x. It is equivalent to require that F is equal to $\mathcal{X}^{(1)}(C)$. Note that each F_c includes $\mathcal{X}^{(1)}(C)$ as a subset. Therefore, $F = \bigcup_c F_c$ is equal to $\mathcal{X}^{(1)}(C)$ if and only if for all c $F_c = \mathcal{X}^{(1)}(C)$. Since these are linear subspaces then we can compare their dimensions.

$\mathcal{X}^{(1)}(C)$ is spanned by all x_i and the constant-1 function:

$$\dim \mathcal{X}^{(1)}(C) = N + 1; \tag{3}$$

The constant-1 function always belongs to $\mathcal{F}^{(1)}(C)$ and to any of the F_c. The subspace of functions that are constant on the restriction can be obtained by adding the constant-1 function to the subspace of functions that are equal to zero on the restriction:

$$F_c = \ker L_c \oplus \{\mathbf{0}, \mathbf{1}\}, \tag{4}$$

$$\dim F_c = \dim \ker L_c + 1. \tag{5}$$

By comparing the dimensions obtained in Equations 3,5 we prove the proposition. □

The algorithm operates on functions using their truth tables. The truth tables are obtained by evaluating the circuit on all possible inputs and recording the values computed in each node. The set of computed truth tables corresponds to $\mathcal{F}(C)$. By removing redundant vectors we can compute a basis $\mathcal{B}$ of $\mathcal{F}^{(1)}(C)$ (and also ensure presence of the constant-1 vector). Then, for each c we take the part of each basis vector that corresponds to the fixed $x = c$ (and r taking all possible values). These parts form the subspace $\operatorname{Im} L_c$. We compute a basis $\mathcal{B}_c$ of these parts. Finally, we verify that

$$\dim \ker L_c = \dim \mathcal{F}^{(1)}(C) - \dim \operatorname{Im} L_c = |\mathcal{B}| - |\mathcal{B}_c| = N. \tag{6}$$

The algorithm is implemented in SageMath [31] and publicly available in [6].

Algorithm 2 Verification of Algebraic Security

```
Input: a Boolean circuit $C(x,r) : \mathbb{F}_2^N \times \mathbb{F}_2^R \to \mathbb{F}_2^M$;
Output: Secure if the circuit $C$ is $\varepsilon$-1-AS for some $\varepsilon < 0.5$,
        Insecure otherwise.

evaluate $C$ on all possible inputs;
associate the vector of computed values to each node of $C$;
let $\mathcal{V}$ be the set of all associated vectors;
let $\mathcal{B}$ be a basis of $\mathcal{V}^{(1)}$;
for all $c \in \mathbb{F}_2^N$ do
    let $\mathcal{V}_c$ be the set of all vectors from $\mathcal{B}$ restricted to the case of $x = c$;
    let $\mathcal{B}_c$ be a basis of $\mathcal{V}_c^{(1)}$;
    if $|\mathcal{B}| - |\mathcal{B}_c| \neq N$ then
        return Insecure;
return Secure.
```

Complexity analysis. The truth tables have size 2^{N+R} bits. Computing the basis of $\mathcal{F}^{(1)}(C)$ takes time $\mathcal{O}(min(2^{N+R}, |C|)^{\omega})$. The same holds for $\operatorname{Im} L_c$ except that the vectors have size 2^R and for small R this can be done more efficiently. The total complexity is $\mathcal{O}(min(2^{N+R}, |C|)^{\omega} + 2^N min(2^R, |C|)^{\omega})$. Recall that by proposition 1 we should consider only the nonlinear nodes of the circuit.

5.5 Algebraically Secure Gadgets

In this section we develop an algebraically secure masking scheme. First we give a broad definition of a masking scheme which we will use further. Then we describe concrete circuits which can be verified to be first-order algebraically secure gadgets using Algorithm 2.

Definition 6 (Masking Scheme). *An N-bit masking scheme is defined by an encoding function $Encode : \mathbb{F}_2 \times \mathbb{F}_2^R \to \mathbb{F}_2^N$, a decoding function $Decode : \mathbb{F}_2^N \to \mathbb{F}_2$ and a set of triplets $\{(\diamond, Eval_\diamond, C_\diamond), \ldots\}$ where each triplet consists of:*

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

For any $r \in \mathbb{F}_2^R$ and any $x \in \mathbb{F}_2$ it must hold that $Decode(Encode(x,r)) = x$. Moreover, the following equation must be satisfied for all operators $\diamond$ and all values $r' \in \mathbb{F}_2^{R'}, x_1 \in \mathbb{F}_2^N, x_2 \in \mathbb{F}_2^N$:

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

The degree of the masking scheme is the algebraic degree of the Decode function. The masking scheme is called nonlinear if its degree is greater than 1.

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

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 the product of the two input bits. Unfortunately, this decoding function is vulnerable to 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 and the hidden value is leaked. 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.

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 $Decode(a, b, c) = 1$ does not imply $a = 1$ or $b = 1$. In fact, such $Decode$ is balanced: both 0 and 1 have exactly 4 preimages. We first describe an insecure yet simple masking scheme based on this decoding function in Figure 1. It is easy to verify that $Eval_{XOR}$ and $Eval_{AND}$ satisfy the requirements from Definition 6. In addition, $Refresh(a, r)$ returns fresh random encoding of a, meaning that $Decode(a) = Decode(Refresh(a, r))$ for any r and new encoding reveals no information about the old encoding.

Fig. 1: An Insecure Quadratic Masking Scheme.

$$\begin{aligned}
Encode(x, r_a, r_b) &= (r_a, r_b, r_a r_b \oplus x), && (7)\\
Decode(a, b, c) &= ab \oplus c, && (8)\\
Eval_{XOR}((a, b, c), (d, e, f)) &= (a \oplus d,\ b \oplus e,\ ae \oplus bd \oplus c \oplus f), && (9)\\
Eval_{AND}((a, b, c), (d, e, f)) &= (ae,\ bd,\ (cd)e \oplus a(bf) \oplus cf), && (10)\\
Refresh((a, b, c), (r_a, r_b)) &= (a \oplus r_a,\ b \oplus r_b,\ c \oplus r_a b \oplus r_b a \oplus r_a r_b). && (11)
\end{aligned}$$

We now observe that $Refresh$ is not ε-1-AS for any $\varepsilon < 1/2$: the computed term $r_a b$ is constant when b is fixed to 0 and equals to r_a otherwise (leading to $\varepsilon = 1/2$). This can be fixed by using an extra random bit r_c to mask a, b through the computations:

$$\begin{aligned}
&Refresh((a, b, c), (r_a, r_b, r_c)) = \\
&\quad \big(a \oplus r_a,\ b \oplus r_b,\ c \oplus r_a(b \oplus r_c) \oplus r_b(a \oplus r_c) \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c\big). \qquad (12)
\end{aligned}$$

The new $Refresh$ function can be verified to be secure using the algorithm from Section 5.4. Moreover, the circuit computing $Eval_{XOR}$ applied to refreshed inputs is secure as well. However, $Eval_{AND}$ is not secure even if composed with the fixed $Refresh$ gadget. Consider the linear combination of computed terms $a(bf) \oplus cf = (ab \oplus c)f$. Here the variables are refreshed masks and can not be fixed by the adversary. However, the refreshing function does not change the hidden value. Therefore, $ab \oplus c$ would be equal to the value hidden by initial non-refreshed shares which can be fixed. Fixing the hidden value to 0 makes the combination $f(ab \oplus c)$ equal to 0 and be equal to the random share f when the hidden value is fixed to 1. We observe that it is possible to use a trick similar to the one used to fix the $Refresh$ function. In fact, the extra random shares added to fix the $Refresh$ function may be reused to fix the $Eval_{AND}$ function. As a result, we obtain a fully secure masking scheme. The complete description is given in Algorithm 3.

Algorithm 3 Minimalist Quadratic Masking Scheme.

```
function ENCODE($x, r_a, r_b$)
    return $(r_a, r_b, r_a r_b \oplus x)$

function DECODE($a, b, c$)
    return $ab \oplus c$

function EVALXOR($(a, b, c), (d, e, f), (r_a, r_b, r_c), (r_d, r_e, r_f)$)
    $(a, b, c) \leftarrow$ REFRESH($(a, b, c), (r_a, r_b, r_c)$)
    $(d, e, f) \leftarrow$ REFRESH($(d, e, f), (r_d, r_e, r_f)$)
    $x \leftarrow a \oplus d$
    $y \leftarrow b \oplus e$
    $z \leftarrow c \oplus f \oplus ae \oplus bd$
    return $(x, y, z)$

function EVALAND($(a, b, c), (d, e, f), (r_a, r_b, r_c), (r_d, r_e, r_f)$)
    $(a, b, c) \leftarrow$ REFRESH($(a, b, c), (r_a, r_b, r_c)$)
    $(d, e, f) \leftarrow$ REFRESH($(d, e, f), (r_d, r_e, r_f)$)
    $m_a \leftarrow bf \oplus r_c e$
    $m_d \leftarrow ce \oplus r_f b$
    $x \leftarrow ae \oplus r_f$
    $y \leftarrow bd \oplus r_c$
    $z \leftarrow a m_a \oplus d m_d \oplus r_c r_f \oplus cf$
    return $(x, y, z)$

function REFRESH($(a, b, c), (r_a, r_b, r_c)$)
    $m_a \leftarrow r_a \cdot (b \oplus r_c)$
    $m_b \leftarrow r_b \cdot (a \oplus r_c)$
    $r_c \leftarrow m_a \oplus m_b \oplus (r_a \oplus r_c)(r_b \oplus r_c) \oplus r_c$
    $a \leftarrow a \oplus r_a$
    $b \leftarrow b \oplus r_b$
    $c \leftarrow c \oplus r_c$
    return $(a, b, c)$
```

Security. First, we verify $Eval_{XOR}$ and $Eval_{AND}$ gadgets using Algorithm 2. We obtain that they are ε-1-AS circuits for some $\varepsilon < 1/2$. Then we construct the ANFs of intermediate functions. The maximum degree is equal to 4. It is achieved for example in the term cf in the gadget $Eval_{AND}$: its ANF contains the term $r_a r_b r_d r_e$. Therefore, $Eval_{AND}$ is ε-1-AS with $\varepsilon \leq 1/2 - 2^{-4} = 7/16$. The gadget $Eval_{XOR}$ has degree 2 and is 1/4-1-AS. Unfortunately, we do not have a pen-and-paper proof for security of the gadgets and rely solely on the verification algorithm (which is able to spot the described weaknesses in the insecure versions of the gadgets).

Verifying security of the encoding function *Encode* can be done in the same way. Clearly, no linear combination of $r_a, r_b, r_a r_b \oplus x$ is constant for any fixed x. The coordinate $r_a r_b \oplus x$ has degree 2 and its weight and bias are equal to 1/4. Therefore, *Encode* is an ε-1-AS encoding with $\varepsilon = 1/4$.

By applying Proposition 3, we obtain that for any adversary $\mathcal{A}$, for any circuit C build from the gadgets $Eval_{XOR}, Eval_{AND}$ and for the described *Encode* encoding we have:

$$\mathsf{Adv}^{\mathsf{PS}}_{C,E,d}[\mathcal{A}] \leq min(2^{Q-R_C}, 2^{-eQ}), \tag{13}$$

where $e = -\log_2 (1/2 + 7/16) \approx 0.093$. According to Corollary 1, in order to achieve provable 80-bit security we need to have $R_C \geq 80(1 + 1/e) \approx 940$ random bits in the circuit. Note that it does not depend on the actual size of the circuit, i.e. 940 random bits are enough for an arbitrary-sized circuit. However, the adversary should not be able to shrink the window so that it contains less than 940 random bits. This is expected to be guaranteed by a structure hiding protection. Finally, we remark that the bounds are rather loose and more fine-grained analysis should improve the bound significantly.

5.6 Implementation

We applied our masking scheme to an AES-128 implementation to estimate the overhead. Our reference AES circuit contains 31,783 gates. It is based on Canright's S-Box implementation [11] and naive implementation of MixColumns. After applying our nonlinear masking scheme and a first-order linear masking scheme on top the circuit expands to 2,588,743 gates of which 409,664 gates are special gates modeling external random bits. The circuit can be encoded in 16.5 MB. Extra RAM needed for computations is less than 1KB. On a common laptop it takes 0.05 seconds to encrypt 1 block. Since the implementation is bitwise, 64 blocks can be done in parallel at the same time on 64-bit platforms. There is still a large room for optimizations. We used the Daredevil CPA tool [24] to test our implementation. Due to the first-order linear masking on top we did not detect any leakage. Pure nonlinear masking scheme does leak the key so the combination of both is needed as we suggested in Section 4. The implementation code is publicly available [6]. We remark that it is a proof-of-concept and not a secure white-box implementation; it can be broken in various ways.

6 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 obtained requirements for a masking scheme to be useful. We divided the requirements into value hiding and structure hiding protections. Furthermore, we suggested that value hiding may be achieved using an algebraically secure nonlinear masking scheme and a classic linear masking scheme. We developed a framework for provable security against the algebraic attack and proposed a concrete provably secure first-order masking scheme. Therefore, a value hiding protection can be implemented.

We believe that our work opens new promising directions in obfuscation and white-box design. In this paper we focused on value hiding protection and developed a first-order protection against the algebraic attack. The natural open question is developing higher-order countermeasures for the algebraic attack. Another direction is to study structure hiding countermeasures. Finally, it seems that pseudorandom generators play an important role in white-box obfuscation and are useful at all layers of protection. Randomness helps to develop formal security models and pseudorandom generators bridge the gap between theoretical constructions and real world implementations. Therefore, designing an easy-to-obfuscate pseudorandom generators is another important open problem.

References

1. S. Banik, A. Bogdanov, T. Isobe, and M. Jepsen. Analysis of Software Countermeasures for Whitebox Encryption. *IACR Transactions on Symmetric Cryptology*, 2017(1):307–328, Mar. 2017.
2. O. Billet, H. Gilbert, and C. Ech-Chatbi. Cryptanalysis of a White Box AES Implementation. In H. Handschuh and M. A. Hasan, editors, *Selected Areas in Cryptography: 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, pages 227–240, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
3. A. Biryukov, C. Bouillaguet, and D. Khovratovich. Cryptographic Schemes Based on the ASASA Structure: Black-Box, White-Box, and Public-Key. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 63–84, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
4. 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.
5. 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.
6. A. Biryukov and A. Udovenko. White-box Tools, 2018. `https://github.com/cryptolu/whitebox`.
7. 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.

8. L. Both and A. May. Decoding Linear Codes with High Error Rate and Its Impact for LPN Security. In T. Lange and R. Steinwandt, editors, *Post-Quantum Cryptography*, pages 25–46, Cham, 2018. Springer International Publishing.
9. P. Bottinelli and J. W. Bos. Computational aspects of correlation power analysis. *Journal of Cryptographic Engineering*, 7(3):167–181, Sep 2017.
10. 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`.
11. D. Canright. A Very Compact S-Box for AES. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 441–455, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
12. C. Carlet. *Boolean Functions for Cryptography and Error-Correcting Codes*, pages 257–397. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2010.
13. B. Carmer, A. J. Malozemoff, and M. Raykova. 5Gen-C: Multi-input Functional Encryption and Program Obfuscation for Arithmetic Circuits. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 747–764, New York, NY, USA, 2017. ACM.
14. S. Chow, P. Eisen, H. Johnson, and P. C. van Oorschot. A White-Box DES Implementation for DRM Applications. In J. Feigenbaum, editor, *Digital Rights Management: ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002. Revised Papers*, pages 1–15, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
15. 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.
16. Y. De Mulder, B. Wyseur, and B. Preneel. Cryptanalysis of a Perturbated White-Box AES Implementation. In G. Gong and K. C. Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010: 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, pages 292–310, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
17. ECRYPT-CSA consortium. CHES 2017 Capture The Flag Challenge. The WhibOx Contest, 2017. `http://whibox.cr.yp.to/`.
18. A. Esser, R. Kübler, and A. May. LPN Decoded. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 486–514, Cham, 2017. Springer International Publishing.
19. P. J. S. G. Ferreira, B. Jesus, J. Vieira, and A. J. Pinho. The rank of random binary matrices and distributed storage applications. *IEEE Communications Letters*, 17(1):151–154, January 2013.
20. 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.

21. H. Gilbert, J. Plût, and J. Treger. Key-Recovery Attack on the ASASA Cryptosystem with Expanding S-Boxes. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 475–490, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
22. 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`.
23. L. Goubin, P. Paillier, M. Rivain, and J. Wang. How to reveal the secrets of an obscure white-box implementation. Cryptology ePrint Archive, Report 2018/098, 2018. `https://eprint.iacr.org/2018/098`.
24. C. Hubain, J. Bos, M. Eder, P. Bottinelli, P. Teuwen, V. H. Le, and W. Michiels. Side-Channel Marvels, 2016. `https://github.com/SideChannelMarvels`.
25. Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Private Circuits II: Keeping Secrets in Tamperable Circuits. In S. Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006. Proceedings*, pages 308–327, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
26. Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings*, pages 463–481, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
27. 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.
28. 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`.
29. 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.
30. P. Sasdrich, A. Moradi, and T. Güneysu. White-Box Cryptography in the Gray Box. In *Revised Selected Papers of the 23rd International Conference on Fast Software Encryption - Volume 9783*, FSE 2016, pages 185–203, New York, NY, USA, 2016. Springer-Verlag New York, Inc.
31. The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 7.3)*, 2016. `http://www.sagemath.org`.
32. M. J. Warrens et al. *Similarity coefficients for binary data: properties of coefficients, coefficient matrices, multi-way metrics and multivariate coefficients*. Psychometrics and Research Methodology Group, Leiden University Institute for Psychological Research, Faculty of Social Sciences, Leiden University, 2008.
33. 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.