Enhanced Encodings for White-Box Designs

Alberto Battistello[1], Laurent Castelnovi[2(✉)], and Thomas Chabrier[2]

[1] Security Pattern, Brescia, Italy
a.battistello@securitypattern.com
[2] IDEMIA, Cryptography and Security Group, Pessac, France
{laurent.castelnovi,thomas.chabrier}@idemia.com

Abstract. Designing a robust white-box implementation against state-of-the-art algebraic and differential computational analysis attacks is a challenging problem. The study of white-box security was revamped by recent advances involving grey box attacks. Since then, many authors have struggled to protect implementations against such new attacks. New designs as well as new security notions appeared, and white-box research in general seems to have greatly benefited from such advances. The current research aims at finding the best encodings and masking schemes to resist tracing attacks. In this perspective we suggest a new encoding scheme that can be applied to white-box designs. By using a modified version of the Benaloh cryptosystem, our design introduces semi-homomorphic properties to the encoding. To the best of our knowledge, this is the first time such properties are applied to an encoding design. This allows reducing the memory requirements and providing a better resistance against tracing attacks. Our encoding is versatile and can be adapted to different ciphers, and in most cases it provides performance improvements with respect to the state-of-the-art.

Keywords: White-Box · AES · Homomorphic cryptosystem · Benaloh cryptosystem

1 Introduction

The mass adoption of connected devices, like smartphones, tablets or smartwatches, implied a deep change in the industry. From basic cellular phones, mobile devices evolved into indispensable microcomputers of everyday life. Our smartphone collects our information, verifies our identity, secures our credit card transactions, replaces our car keys, enables us to watch movies and series, and can perform many other "useful" operations.

It is thus mandatory that such smart objects provide users with enough security for the collected data. This turns out to be the role of trusted execution

A. Battistello—Part of this work was done while the first author was working at IDEMIA.

V. Grosso and T. Pöppelmann (Eds.): CARDIS 2021, LNCS 13173, pp. 254–274, 2022.
https://doi.org/10.1007/978-3-030-97348-3_14

environments (TEE) that solved the security problem by delegating the sensitive tasks to an embedded secure element (eSE). However, smartphones are far from being standardized in all their aspects. For example the same OS version can run on multiple platforms. These platforms may have different flavors of eSEs or none at all. Thus, applications must adapt the security to the device at hand. Far from being an easy task, providers must even protect the device from the legitimate user itself, which in some cases may behave wickedly.

The white-box model takes on its full meaning in such an environment, where the cryptographic implementation is exposed. The white-box model can be seen as the opposite of the black box model, where the attacker has only access to the inputs and the outputs. Indeed, the white-box model assumes that the cryptographic primitive runs in an untrusted environment. In such a scenario the attacker has full access to the device: he can access the binary code of the application and observe or interact with the device's execution, in order for example to extract the encryption key from the implementation. It is thus very difficult to provide efficient security solutions against such threats. Nonetheless, due to the mass adoption of smart-devices, solutions are required by the industry to mitigate the problem.

Besides, the market needs white-box solutions for DRMs, Pay-TV, secure storage, etc., motivating researchers all over the world to keep working on it. The competitive spirit of researchers, together with the market pressure, also motivated the creation of international challenges, like the WhiBox contest [1,2], where users are invited to submit white-box AES implementations, and each participant can try to break others' submissions. In these challenges, a white-box is considered unbroken until its key is found or its functionality reversed. Points are assigned to the participants who implement the longer lasting unbroken white-box and to those breaking the strongest ones. For example, an interesting result of the 2019 edition of the WhiBox competition [2] is that the best white-boxes withstood attacks for more than one month. Thus, despite theoretically flawed, the security offered by actual implementations can be sufficient for content which value is limited to a short-term period (like for example a live football world cup).

One of the many techniques used in the contest of white-box is the use of encodings [20], affine or non-linear functions applied to the input/output of tables, and unknown by the attacker. These random encodings provide a map from the clear world and the encoded world, and allow randomization of key-dependent data. Such countermeasures are however expensive to deploy. The designer often needs to find a tradeoff between security and complexity (as a combination of memory requirements and running time).

The purpose of our work is thus to suggest an encoding principle that can help to reduce such complexity rise, while maintaining the security at the state-of-the-art. In particular, we suggest a novel way to create encodings that are both cheap and safe. We show how to use a degraded semi-homomorphic encryption scheme, based on the work of Benaloh [6] to build non-linear encodings that provide better security against algebraic attacks as well as security against differential computation analysis (DCA). Our encodings allow dropping the use of

tables for the most used operations in crytographic algorithms. Also, our proposition provides better performances for most operations by exchanging lookup table accesses with CPU operations. We provide both a security evaluation of our proposition together with an application to an AES-128 white-box. Based on our suggested AES-128 white-box, we provide memory and performances comparisons against state-of-the-art implementations.

This paper is organized as follows. Section 2 is dedicated to the state-of-the-art in white-box designs and attacks. Afterwards, Sect. 3 introduces our new encoding scheme. In Sect. 4 we provide a security analysis of our suggested white-box encoding. The complexity in terms of memory and time of our new design is evaluated in Sect. 5. Finally some suggestions for further developments of our idea are provided in Sect. 6. Section 7 concludes this work.

2 State-of-the-Art

In 2002, two seminal papers from Chow *et al.* [19,20] introduced a new way to implement cryptographic algorithms that provided some risk mitigation. Such solutions to resist white-box attacks are known as "white-box cryptography" (WBC). They paved the way for a new and very active research field for both theoreticians and practitioners.

2.1 White-Box Designs

The work of Chow *et al.* [19,20] exposed the first white-box descriptions for both DES and AES ciphers. Their design has been the white-box implementation reference since then. Chow *et al.* introduced the use of tables to perform computations. Also, similar to the randomization technique of Kilian [31], by *encoding* each table with input/output functions unknown to the attacker, Chow *et al.* provided an initial solution to the problem of obfuscating a program.

After the first white-box propositions were published, Link *et al.* suggested an improved version of the DES white-box in [36], to better resist attacks. However, several attacks that allowed to recover the key hidden in the DES white-box were proposed by Wyseur *et al.* [47] and by Goubin *et al.* [27]. In parallel, the AES white-box was broken using an algebraic attack by Billet *et al.* [7], which was further refined and generalized by Michiels *et al.* [39]. In order to thwart such attacks, other authors proposed further white-box designs of the AES-128, for example Bringer *et al.* in [16] suggested an approach based on polynomials, but their suggestion was broken by De Mulder *et al.* in [23]. In 2009, Xiao and Lai proposed an improvement of Chow *et al.* AES white-box in [48], but again, their proposal was broken by De Mulder *et al.* [22]. In 2010, Karroumi suggested in [30] to use dual ciphers to protect the AES white-box. Unfortunately such a design was also broken by Lepoint *et al.* [35].

Generally speaking, all propositions of AES and DES white-box designs have been shown to be theoretically broken in the sense that the embedded key can be

extracted. For each proposition a theoretical attack was found, see for example [7, 35, 39].

In a parallel thread of work, researchers tried to clarify the security notions related to the white-box context, thus works like those of Delerablée *et al.* [24], Saxena *et al.* [44] and Bock *et al.* [12] appeared.

A further step in the understanding of the security of white box implementations was brought forward by the introduction, in 2015 and 2016, of two attacks borrowed from the field of physical security. These attacks were used to break AES white-box suggestions appeared meanwhile, like the work of Luo *et al.* [37] and Lee *et al.* [32]. Such attacks were fault injection attacks, presented by Sanfelix *et al.* [42] and side-channel analysis (a.k.a. differential computation analysis, DCA for short), by Bos *et al.* [14]. Eventually such attacks provided easier methods to break all previous contributions, and researchers started to shift their interest from the algebraic security, to such a physical security dimension.

Indeed, several practical attacks have further reduced the security margin provided by a white-box implementation [14, 42]. However, all is not lost, as such advanced attacks motivated the study of advanced countermeasures. During the WhiBox contest editions, for example, a few implementations stood more than one month, while the hacking community tried to break them. From such implementations stemmed new understanding and improved countermeasures (see for example [9, 13, 28, 41, 43]), that allowed to thwart, or mitigate, the attacks explained so far.

2.2 White-Box Encoding

In order to counteract such new attacks, designers suggested to adapt known embedded security countermeasures, like the masking countermeasure [29, 40]. For example the work of Lee *et al.* [34] suggested a masked AES white-box implementation. Although providing an undeniable improvement on the security of white-box instances, such countermeasures deteriorate white-box implementations in terms of memory and performances.

Encodings are one of the key concepts introduced by Chow *et al.* Despite their use in [20] to counteract algebraic attacks authors worked in the recent publications to improve the effectiveness of encodings and to provide a masking stage to the algorithm. The two notions of encodings and masking are sometimes overlapping, and an encoding scheme may act as a masking scheme, and vice-versa. Loosely speaking, masking is a technique that removes the correlation between a value and its representation by, for example, using Shamir's secret sharing [46], while encoding is the application of (secret) input and output bijections to a transformation. Thus a secret sharing scheme can be seen as the application of the XOR bijection with a mask to the identity transformation, while an encoding can be interpreted as the application of some (non) linear masking scheme like for example [17] to a secret value.

It has been shown by various authors [9, 13, 28, 41, 43] how an accurate choice of encoding is paramount to the security of the white-box. In particular it seems that the best approach, as suggested by recent works [9, 28, 33, 45] is to use a

linear masking on top of a non linear one. Such countermeasures are however expensive to deploy. The designer often needs to find a tradeoff between security and complexity (as a combination of memory requirements and running time). As an example, the winner of the 2019 edition of the WhiBox competition [2] used such an encoding (as revealed by the reverse engineering attack by Goubin *et al.* [28]) and the smallest implementation was about 20 MB for an AES-128 encryption.

In the following section we suggest a new encoding scheme that provides an improved security, fast operations, and a reduced memory footprint.

3 New Encoding Design for White-Box Constructions

Our proposal is directly inspired by the Benaloh cryptosystem, suggested in [6]. The original scheme is partially homomorphic, meaning that it allows to perform only one type of operation on plaintexts in the cipher domain. We modified the original Benaloh cryptosystem while preserving the semi-homomorphic properties in order to provide a new encoding scheme. Below, we recall the mathematical background that is used in the rest of the paper. Afterwards, we provide a brief explanation of the Benaloh cryptosystem together with our suggested modifications to use it as a white-box encoding.

3.1 Preliminaries

In the rest of this paper we will use notions such as *quadratic residue*, or *higher residue*. Such notions are detailed below.

Definition 1 (Quadratic Residue). *Let $m \in \mathbb{Z}_n^*$ for an odd integer n. Then m is a* quadratic residue *modulo n if there exists $x \in \mathbb{Z}_n^*$ such that:*

$$x^2 \equiv m \bmod n.$$

If no such x exists, then m is a quadratic non-residue *modulo n.*

Definition 2 (Legendre symbol). *Given a prime p and $m \in \mathbb{Z}_p^*$, the* Legendre symbol *of m modulo p is denoted $\left(\frac{m}{p}\right)$ and defined as follows:*

$$\left(\frac{m}{p}\right) = \begin{cases} 0 & if m \equiv 0 \mod p, \\ 1 & if\ m\ is\ a \text{ quadratic residue } modulo\ p, \\ -1 & otherwise. \end{cases}$$

Definition 3 (Quadratic Residuosity Problem). *The* quadratic residuosity problem *(QRP) is the following: given an odd composite integer n and $m \in \mathbb{Z}_n^*$, decide whether m is a* quadratic residue *or a* quadratic non-residue *modulo n.*

Remark 1. If n is prime, then the QRP can easily be solved by Euler's criterion: for any $m \in \mathbb{Z}_n^*$, $\left(\frac{m}{n}\right) \equiv m^{(n-1)/2} \mod n$ (see for instance [38]).

Definition 4 (Higher Residue). *Let $m \in \mathbb{Z}_n^*$ for an odd integer n, m is said to be a* d-residue *modulo n if there exists $x \in \mathbb{Z}_n^*$ such that*

$$x^d \equiv m \bmod n.$$

If no such x exists, then m is said to be a d-non-residue *modulo n.*

Definition 5 (Higher Residuosity Problem). *The* higher residuosity problem *(HRP) is the following: given an odd composite integer n and $m \in \mathbb{Z}_n^*$, decide whether m is a* d-residue *or a* d-non-residue *modulo n.*

Remark 2. If the factorization of n is known, then the HRP can easily be solved (see [38]).

3.2 Original Description of Benaloh Cryptosystem

The Benaloh cryptosystem, introduced by Benaloh in 1994 [6] and improved by Fousse *et al.* [25], is an extension of the Goldwasser-Micali cryptosystem (GM) [26]. The latter's security relies on the QRP, while the former's on the HRP. Where the GM cryptosystem encrypts bits individually, Benaloh's improvement allows blocks of bits to be encrypted at once. Both schemes are probabilistic cryptosystems in the sense that several encryptions of the same message under the same key yield different ciphertexts. In this section, we describe the original Benaloh's cryptosystem.

Key Generation. The public and private key are generated as follows.

- Choose a block size r and two large prime numbers p and q such that:
  - $r|(p-1)$,
  - $\gcd(r, (p-1)/r) = 1$,
  - $\gcd(r, q-1) = 1$.
- Set $n = p \times q$ and compute $\phi(n) = (p-1)(q-1)$.
- Select $y \in \mathbb{Z}_n^*$ such that, for any prime factor r_i of r:
  - $y^{\phi(n)/r_i} \not\equiv 1 \bmod n$.

The public key is (n, r, y), and the private key is (p, q).

Encryption. Given the public parameters (n, r, y) and a an element of $\mathbb{Z}_r$, the encryption E_r is defined as:

$$E_r(a) = y^a u^r \bmod n,$$

where u is a random number in $\mathbb{Z}_n^*$.

Decryption. Given decryption key (p, q), and ciphertext c, the decryption D_r is defined as:

$$D_r(c) = \log_x(c^{\phi(n)/r}) \bmod n,$$

where $x = y^{\phi(n)/r} \bmod n$.

The homomorphic property is easily verified:

$$\begin{aligned} E_r(a) \times E_r(b) &\equiv y^a u_0^r \times y^b u_1^r \mod n \\ &\equiv y^{a+b}(u_0 u_1)^r \mod n \\ &\equiv E_r(a+b) \mod n. \end{aligned}$$

Our work aims at using this cryptosystem as an encoding. This allows to homomorphically perform some operations on the encoded values and thus reduce the overall memory cost of the white-box. In the following, we propose some modifications to achieve our goal.

3.3 Modified Benaloh Cryptosystem

We describe and motivate in this section our adaptations of the Benaloh cryptosystem to make it suitable for using as a white-box encoding. We deal with the encoding itself in Sect. 3.4.

Key Generation. The public and private key are generated as follows.

- Choose a prime number p.
- Choose a block size $r = 2^k$ such that $k \geq 2$ and r is the highest power of 2 which divides $p - 1$.
- Select randomly y a generator of $\mathbb{Z}_p^*$.
- Select randomly $t \in \mathbb{Z}_p^*$.

The public key is (p, r) and the private key is (t, y).

Compared to the original key generation, the private key t is introduced, u is fixed to 1 and the modulus is a prime number instead of a composite of two prime numbers. For the sake of simplicity, we keep the expression "private key" despite the fact that we use our private key both for encryption and decryption.

Encryption. Given the private key (t, y) and the public parameter p, the encryption E_t is defined as:

$$E_t(m) = t\, y^m \bmod p, \tag{1}$$

where m is an element of $\mathbb{Z}_p$.

It is easily verified that:

$$E_t(m_0) \times E_t(m_1) \equiv E_{t^2}(m_0 + m_1) \mod p.$$

Compared to the original encryption, the definition set of m is extended to the entire group $\mathbb{Z}_p$.

Decryption. Given the decryption key (y, t, p, r) and the ciphertext c, the decryption function D_t is defined as:

$$D_t(c) = \log_x((t^{-1}c)^{(p-1)/r}) \bmod p, \tag{2}$$

where $x = y^{(p-1)/r} \bmod p$.

The decryption differs from the original one only by the multiplication by $t^{-1} \bmod p$.

Motivations

About the Modulus. The modulus has been chosen to be a prime number. Since our proposal will rely on the QRP and will use a small modulus, there is no security benefit in choosing a composite modulus as the QRP is easy to solve even for small composite moduli.

About the Block Size. The block size r has been chosen to be a power of 2 in order to ensure that the least significant bit (LSB) of $D_t(E_t(a + b))$ equals the exclusive-or between the LSB of a and the LSB of b. Our proposal is based on this property, with the condition $r = 2^k$ and $k \geq 2$.

About the Base. The base y has been chosen to be a generator of $\mathbb{Z}_p^*$ as it guarantees that $y^{(p-1)/r} \not\equiv 1 \bmod p$. It is a requirement from the original key generation algorithm. The secrecy of y is a consequence of Sect. 4.1.

About the Key. In our proposal, the senstive data bits will be carried by the LSB of the exponent of y. The private key t has been introduced to hide this bit. If t was not present, with only the three previous modifications to the original Benaloh scheme, the sensitive bit m could be guessed by using the Legendre symbol of $E_1(m)$, which equals 1–2 m. The multiplication of $E_1(m)$ by a uniformly random number t makes its Legendre symbol equal to $\left(\frac{t}{p}\right)(1 - 2m)$, which is equal to 1 or -1 with the same probability 1/2.

3.4 Modified Benaloh Cryptosystem as White-Box Encoding

In this section we show how our modified Benaloh cryptosystem can be used as a white-box encoding. Our proposal relies on three well-known facts:

1. Any Boolean function can be expressed as a logical circuit composed of `XOR`, `AND` and `NOT` gates.
2. The sum in $\mathbb{Z}$ of two bits a and b is $a+b = (a \wedge b) \,||\, (a \oplus b)$, where $||$ denotes the concatenation operator.
3. The sum in $\mathbb{Z}$ of one bit a with 1 is $a + 1 = a \,||\, (a \oplus 1) = a \,||\, \bar{a}$.

We thus propose to consider the cryptographic algorithm as a logical circuit which gates are modified-Benaloh encoded. We describe hereafter how to encode (resp. decode) the circuit's input (resp. output) and how to evaluate its gates.

Encoding and Decoding Functions

Encoding Step. To encode a bit a, the modified key generation method (see Sect. 3.3) is run and the modified encrypting function (cf. Eq. (1)) is applied to $2s + a$, where $0 \leq s < (p-1)/2$ is uniformly drawn at random. We will denote the encoding function by Enc:

$$\mathsf{Enc}(a) = E_t(2s + a).$$

Here, s is introduced to make our encoding probabilistic, as does the random u in the original Benaloh scheme. Introducing this s instead of keeping the original u, we save one entropy bit: indeed, if the bit x was encoded as $E_t(x) = ty^x u^r \bmod p$, then $E_t(x) = ty^{vr+x} \bmod p$ with $v = \log_y u$, and since $r \geq 4$ and $r \,|\, (p-1)$, $(vr + x \bmod p - 1) \equiv x \bmod 4$. In other words, the second LSB of the exponent of y would always be 0. On the other hand, having both u and s does not provide more entropy to $\mathsf{Enc}(x)$. Therefore, we discard u but introduce s to keep the probabilistic property of the Benaloh scheme.

Decoding Step. To decode a value $\mathsf{Enc}(a)$, the modified decryption function (cf. Eq. (2)) is applied and the result is reduced modulo 2 to get a single bit. We will denote the decoding function by Dec:

$$\mathsf{Dec}(\mathsf{Enc}(a)) = D_t(\mathsf{Enc}(a)) \bmod 2.$$

Evaluating Logical Gates

As recalled before, a circuit can be constructed using only `XOR`, `AND` and `NOT` gates. We describe hereafter how each gate can be evaluated under the modified Benaloh-encoding.

From now, we consider that the encodings of the bits a and b are:

$$\mathsf{Enc}(a) = E_{t_0}(2s + a) = t_0\, y^{2s+a} \bmod p$$
$$\mathsf{Enc}(b) = E_{t_1}(2s' + b) = t_1\, y^{2s'+b} \bmod p.$$

`XOR` *Implementation.* To compute an encoding of $a \oplus b$, it is sufficient to multiply the encoding of the two bits:

$$\begin{aligned} \mathsf{Enc}(a)\mathsf{Enc}(b) &\equiv t_0 t_1\, y^{2(s+s')+a+b} \bmod p \\ &\equiv t_0 t_1\, y^{2(s+s'+ab)+(a \,\oplus\, b)} \bmod p. \end{aligned} \tag{3}$$

Let us verify that indeed $\mathsf{Dec}(\mathsf{Enc}(a)\mathsf{Enc}(b)) = a \oplus b$. Let be $\alpha \equiv 2(s + s' + ab) + (a \oplus b) \bmod p - 1$. Since $p - 1$ is even, $\alpha \bmod 2 = a \oplus b$. Setting $t = t_0 t_1$, the decryption function D_t returns α reduced modulo $r = 2^k$, thus this step preserves the k least significant bits of α. Therefore, the result of the decoding function is $a \oplus b$.

It follows that an arbitrary number of `XOR` gates can be evaluated in a row without caring for carries, that is to say, if $\{a_1, \ldots, a_n\}$ is a set of bits:

$$\mathsf{Dec}\left(\prod_{i=1}^{n} \mathsf{Enc}(a_i)\right) = a_1 \oplus \cdots \oplus a_n.$$

It is worth noticing that only modular multiplications, hence only CPU operations, are needed to evaluate `XOR` gates. Thus, the evaluation of any linear function comes at no memory cost by using our modified-Benaloh encodings.

`AND` *Implementation.* Contrary to the `XOR` operation, we use tables to implement the `AND` operation. A naive solution is to use one table with two operands as inputs. This table is used to decode each operand, evaluate the `AND` gate and re-encode the result. Each table admits p^2 entries and returns a $\log_2 p$-bit value, which implies a memory consumption of $p^2\lceil \log_2 p \rceil$ bits per table.

We present instead another solution based on right-shift tables admitting only p entries, thus reducing the memory consumption per table down to $p\lceil \log_2 p \rceil$ bits. This solution is based on the fact that when $\mathsf{Enc}(a)$ and $\mathsf{Enc}(b)$ are multiplied together, y is raised to the power $2(s + s' + ab) + (a \oplus b)$ (see Eq. (3)). Then, right-shifting it results in $s + s' + ab$. Similarly, right-shifting the exponent of y in $\mathsf{Enc}(a)$ (resp. $\mathsf{Enc}(b)$) gives s (resp. s'). By adding these three numbers, we have $2(s + s') + ab$. Thus, we obtain with our solution the encoding of the desired bit ab as $\mathsf{Enc}(ab) = E_t(2s'' + ab)$ for some t and $s'' = s + s'$.

The first step is thus to right-shift the power of y in the expression of $\mathsf{Enc}(a)\mathsf{Enc}(b)$, $\mathsf{Enc}(a)$ and $\mathsf{Enc}(b)$. The fact that $r = 2^k$ impedes these three right shifts to be performed in a homomorphic way by successive left shifts. Thus, they have to be tabulated. Each table *decrypts* its entry (instead of decoding it, otherwise the random s's would be lost), then right-shifts the result and finally *encrypts* the shifted value to which is added a random even number (see Fig. 1).

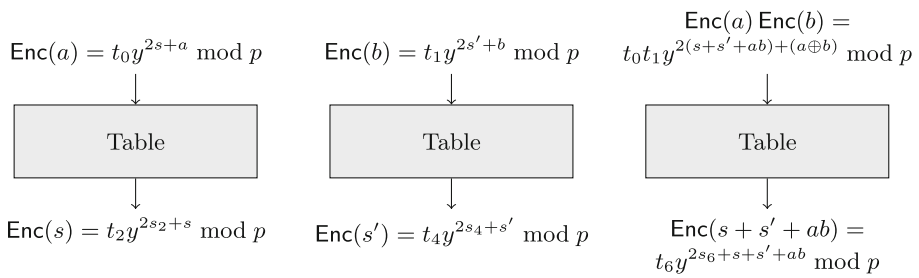

Fig. 1. Input and output of the three right-shift tables.

The output of the three tables are multiplied together to get, according to the notations in Fig. 1, $\mathsf{Enc}(ab) = ty^{2(S+s+s')+ab} \bmod p$ with $t = t_2 t_4 t_6$ and $S = s_2 + s_4 + s_6$.

`NOT` *Implementation.* To get an encoding of $\bar{a} = a \oplus 1$, one must multiply $\mathsf{Enc}(a)$ by any non-quadratic residue modulo p: let v be one of them, then there exists an integer α such that $v \equiv y^{2\alpha+1} \bmod p$. Thus $v\mathsf{Enc}(a) \equiv t_0 y^{2(s+\alpha)+a+1} \equiv t_0 y^{2(s+\alpha+a)+\bar{a}} \bmod p$.

Another solution for the NOT implementation can be used: it is sufficient to integrate the NOT operation to the next table. For instance, the next table can be an AND table. The evaluation of the NOT gate can thus be delegated to the next table due to the linear property of the XOR gate.

From now, sequences of binary operations composed with ANDs, XORs and NOTs can be computed under our modified-Benaloh encoding.

3.5 Using the Modified-Benaloh Encoding in a White-Box Design

Hereafter, we give a way to design a white-boxed cipher by using the modified-Benaloh encoding. As an example, we give in Sect. 5 an AES implementation using our proposal.

Overall Parameters. The modified-Benaloh key generation algorithm (see Sect. 3.3) is run to get the parameters p, r and y. They have to be common to all encoded bits within the white-box, in order for the logical gates to be correctly evaluated as described in Sect. 3.4. These parameters might be refreshed at any moment, at the cost of an extra decoding-then-encoding step on each encoded bit to switch from the former set of parameters to the new one.

Encoding Data Bits

Encoding the Key. The ℓ-bit cipher key is embedded within the white-box in a modified-Benaloh encoded form. The white-box embeds a table with ℓ entries, each one being of the form $t_j\, y^{2s_j+k_j} \bmod p$, where $1 \leq j \leq \ell$ and k_j is the j^{th} key bit.

Encoding the Plaintext. The white-box turns each plaintext bit to a modified-Benaloh encoding using a 1-bit input table. Each bit of the plaintext has its own encoding table. Such tables are of the form:

$$T_i = \{\mathsf{Enc}(0), \mathsf{Enc}(1)\} = \{t_i\, y^{2s_i} \bmod p,\ t_i\, y^{2s'_i+1} \bmod p\}.$$

Note that t_i here is fully independent from the t_j's that protect the cipher key.

Let us detail here the use of the private key t_i. We can remark that it must be the same for the two possible encodings of the same bit: suppose that we could have $\mathsf{Enc}(0) = E_t(2s)$ and $\mathsf{Enc}(1) = E_{t'}(2s'+1)$ with $t \neq t'$ and (t, t') being different for each bit to encode. Let us consider a table that, within the circuit, decrypts its entry. As an example, it could be a table evaluating an AND gate. For this decryption step, the private key of the table entry, or a product of several of them, if the entry is a product of several encodings, has to be known. Then, when the entry of such a table would be a combination of n encoded input bits, 2^n tables (one per possible n-tuple of encoded bits) would be necessary to perform the operation delegated to the table. Therefore, the private key t_i has to be fixed per encoded bit and not per encoding.

Cipher Evaluation. Once each bit of the plaintext is encoded, the cipher, designed as a logical circuit, can be evaluated. One has just to perform binary operations on the encoded bits as explained in Sect. 3.4.

Decoding Data Bits. The ciphertext's bits are decoded from their modified-Benaloh encoded form by a table that applies the decoding function on its entries. Note that one table per ciphertext bit is necessary, as each decoding table uses a different private key from the other decoding tables.

4 Security Considerations

With our modified-Benaloh scheme, we proposed to encode each bit b as:

$$\mathsf{Enc}(b) = t\, y^{2s+b} \mod p$$

for some random numbers t and s. All logical operations between encoded bits are done in the encoding domain with modular multiplications and table accesses.

In this section we provide a security analysis of our proposal, by addressing some potential flaws.

4.1 About the Shift Tables

In this section, we study two attack paths opened by the shift tables used to evaluate the `AND` gates.

The input-output-squared attack. Let $n_{i,b} = t_i\, y^{2s_b+b} \bmod p$ be the Benaloh-encoded input of a shift table and $n_{o,b} = t_o\, y^{2s+s_b} \bmod p$ the corresponding output. For the sake of clarity, we set here $s = 0$ as it will not be useful in this section. Then $n_{i,b}\, n_{o,b}^2 \equiv t_i\, t_o^2\, y^{4s_b+b} \bmod p$.

The attacker can collect $n_{i,0}, n_{o,0}, n_{i,1}$ and $n_{o,1}$ and compute:

$$\begin{aligned} z &= n_{i,0}\, n_{i,1}\, n_{o,0}^2\, n_{o,1}^2 \bmod p \\ &= t_i^2\, t_o^4\, y^{4(s_0+s_1)+1} \bmod p. \end{aligned}$$

Since y is a generator of $\mathbb{Z}_p^*$, there exists $\tau_i, \tau_o \in \{0, \ldots, p-1\}$ such that $t_i = y^{\tau_i} \bmod p$ and $t_o = y^{\tau_o} \bmod p$. This implies that $z = y^{4(\tau_o+s_0+s_1)+2\tau_i+1} \bmod p$.

If the attacker can guess the least significant bit (LSB) of τ_i from z, then he can compute $\left(\frac{t_i}{p}\right) = \tau_i \bmod 2$ and therefore b. However, whereas the LSB of $\log_y z$ is actually independent from the chosen generator y, it is not the case of its second LSB: let y' be another generator of $\mathbb{Z}_p^*$ and a be the integer such that $y = y'^{2a+1} \bmod p$; then $\log_{y'}(z) = \log_{y'}(y) \log_y(z) \equiv 2(a + \tau_i) + 1 \bmod 4$. Since a can be odd or even[1], the knowledge of z does not give any information on $\tau_i \bmod 2$.

[1] For instance, in $\mathbb{Z}_{59}^*$, all odd powers of 2 but $2^{57} \equiv -1 \bmod 59$ are generators.

The Frequency Attack. Suppose that we want to AND two bits a_0 and a_1. Let $s_0^{(0)}, s_0^{(1)}, s_1^{(0)}$ and $s_1^{(1)}$ be four integers modulo $(p-1)/2$ and let $\mathsf{Enc}(a_0) = t_0\, y^{2s_0^{(a_0)}+a_0} \bmod p$ and $\mathsf{Enc}(a_1) = t_1\, y^{2s_1^{(a_1)}+a_1} \bmod p$. The table fed by $\mathsf{Enc}(a_0) \times \mathsf{Enc}(a_1)$ during the evaluation of the AND gate (see Fig. 1) returns some $\alpha = t\, y^{2\delta+\beta} \bmod p$ where β is a bit that equals $a_0 a_1 \oplus ((s_0^{(a_0)} + s_1^{(a_1)}) \bmod 2)$. Since $\left(\frac{\alpha}{p}\right) = \left(\frac{t}{p}\right)(1-2\beta)$, $\left(\frac{\alpha}{p}\right) = \left(\frac{t}{p}\right)$ with probability 3/4 if $s_0^{(0)} \equiv s_0^{(1)} \bmod 2$ and $s_1^{(0)} \equiv s_1^{(1)} \bmod 2$, which leaks the value of $a_0 a_1$. Therefore, the LSB of the random $s_0^{(0)}, s_0^{(1)}, s_1^{(0)}$ and $s_1^{(1)}$ should be adjusted to avoid this, for instance by imposing $s_i^{(0)} \bmod 2 = s_i^{(1)} + 1 \bmod 2$ for $i = 0, 1$. In the case where such a solution would not be tractable, we suggest to implement a Boolean-masked AND to decorrelate the content of the shift tables from the bits to AND together.

4.2 About the Key

We recall that in our proposal in Sect. 3.3, the cipher key is embedded in a Benaloh-encoded form within the white-box. Encoded this way, no information can be extracted on the key just looking at its encoded form: since each key bit b has its own random private subkey t, $E_t(b)$ is indistinguishable from a random number in $\mathbb{Z}_p$ and so t acts like a one-time pad.

On the other hand, some information about the key may be revealed by the following DCA-like attack. Seeing the cipher as a logical circuit, any gate output is Benaloh-encoded with a random t unknown from the attacker but fixed over all executions of the white-box. Then any variation of the Legendre symbol of the gate output is only due to a variation of the encoded bit.

Therefore, the attacker can focus on an AND gate which inputs depend on a few key bits. By making an assumption on these key bits, the attacker can compute an expected sequence of outputs of his targeted AND gate when the input plaintexts vary. Then, by comparing this sequence to the Legendre symbols actually output by the white-box, the attacker can accept or reject his hypothesis on the subkey. Repeating this procedure with different gates depending on other key bits, he can reduce the subset of possible cipher keys.

In order to thwart such an attack, a possible countermeasure consists in implementing a Boolean-masked circuit (in addition to applying the Benaloh-encoding). The impact of the masking on the XOR gates comes at no memory cost. However, concerning the AND masking, we suggest to use the secure AND proposed by Biryukov *et al.* [8]. The impact on the number of tables is thus limited to only a factor 4.

4.3 Summary

In order to prevent the identified security issues, we decline our Benaloh-encoded white-box into two flavours:

1. *Proposition 1:* a lightweight white-boxed cipher without countermeasure,
2. *Proposition 2:* a white-boxed cipher implemented as a Boolean-masked circuit to protect vulnerable `AND` gates (Sect. 4.2).

We close this section with a brief estimation of the computational effort needed to defeat our propositions:

1. *Proposition 1* can be defeated by the Legendre symbol attack of Sect. 4.2. It is equivalent to a differential computational analysis (DCA) with a Legendre symbol leakage model. The results summarized in [9, Table 1] imply that the cost of the DCA Legendre symbol attack is $O(nk \log_2 p)$, where n is the length of the trace, k is the number of key hypotheses and $\log_2 p$ the cost of performing Euler's criterion.
2. *Proposition 2* can be defeated by a 2^{nd}-order DCA if the sensitive data are shared into 2 bits. [9, Table 1] implies in this case that the cost of the Legendre symbol attack is $O(n^2 k \log_2 p)$.

5 Performances: Example with AES-128 Encryption

In this section we provide performances estimations for an AES white-box implementation designed with our two propositions with the security improvements described in Sect. 4.3. Furthermore, we compare our AES-128 white-box design performances against other state-of-the-art designs, with respect to execution time and space requirements.

The AES can be written with only elementary gates. In particular, an AES can be only composed of `XOR`, `NOT XOR` (`NXOR`), and `AND` operations. The AES requires only `XOR` gates to implement, except the SubBytes function that requires also `AND` gates. For SubBytes, we use the bitsliced software implementation proposed by Calik [18, Sect. 7] which is an improvement of the Boyar and Peralta [15] circuit. He proposed an AES SBox with 113 gates, composed of 77 `XOR`s, 4 `NXOR`s and 32 `AND`s. Following our proposal in Sect. 3.4), only `AND` gates require tables. Hence, the memory consumption of an AES-128 with our proposition is $15\,360 \times p \times \lceil \log_2(p) \rceil$ bits, where $15\,360 = 32 \times 10 \times 16 \times 3$:

- 32 `AND`s are required for each SBox.
- There are 10 rounds in the AES-128.
- There are 16 input bytes.
- There are 3 tables for each `AND` with our solution (see Sect. 3.4).

Besides, the secret key is considered to be embedded within the white-box in a Benaloh-encoded form: it corresponds to 128 tables of $\lceil \log_2(p) \rceil$ bits. Finally, the white-box requires one table to encode each plaintext bit, and one table

per ciphertext bit to remove all the random masks t_i accumulated through the circuit: it corresponds to $128 \times 2 = 256$ tables of $p\lceil \log_2(p) \rceil$ bits. It leads to a total of $15\,744 = 15\,360 + 128 + 256$ tables. We can note that whatever the size of the chosen parameter p, the number of tables is always the same, i.e. 15 744.

In a same way, the execution time is constant and is not dependent on the size of the parameters (as long as they fit in the architecture registers). Indeed, whatever the chosen parameters of the implementation, we have 15 744 tables to access and $30\,520 = (92 + 128 + ((77 + 4 + 3 \times 32) \times 16)) \times 10$ `XOR` gates to evaluate, where:

- 92 represents the number of `XOR`s in the MixColumns.
- 128 represents the number of `XOR`s for the AddRoundKey.
- $77 + 4$ represents the number of `XOR`s in the SBox computations.
- 3×32 represents the number of `AND`s in the SBox computations, and the number of `XOR`s required during the `AND` calculations.
- 16 represents the number of SBox in the AES-128.
- 10 represents the number of rounds in the AES-128.

It leads to an execution time of 15 744 table accesses and 30 520 short modular multiplications. The `NOT` gates are not taken into consideration. Indeed, we consider that these gates can be delegated to the next table of the circuit.

Table 1 gives the min and max bounds for memory consumption of an AES white-box using our proposition, where the bounds depend on the used prime number p.

Table 1. Memory consumption according to the bit size of p.

$\lceil \log_2(p) \rceil$	Memory consumption (megabytes, MB)
4	[0.07, 0.11]
6	[0.39, 0.74]
8	[2.03, 4.01]
10	[10.10, 20.14]
12	[48.42, 96.77]
14	[225.86, 451.64]
16	[1 032.35, 2 064.61]

In Fig. 2 we provide a comparison of sizes and estimated execution time of published AES-128 white-boxes and our suggestions *Proposition 1* and *Proposition 2*. The entries are sorted by publication date from the left to the right. The comparison aims at providing an overview of the evolution of white-box design sizes according to the execution time of the implementation. Figure 2 thus compares the size of a reference AES implementation [21] with the size of

the white-box implementations of Chow *et al.* [20], Bringer *et al.* [16][2], Xiao *et al.* [48], Karroumi [30], Lee *et al. 1* [32], Lee *et al. 2* [34], Lee *et al. 3* [33], Luo *et al.* [37], Bai *et al.* [3], Biryukov *et al.* [9], Seker *et al.* [45][3] and this work.

The execution time estimations are obtained by using the number of LUT accesses multiplied by 0.8 ns (typical RAM access times for DDR3 memory). For the works that did not use any table, we accounted for 0.5 ns per computation (typical operation time for a 2 GHz processor).

For our implementation, we choose a 6-bit prime number. Given the remarks of Sect. 4.3, a 6-bit prime number does not significantly weaken our white-box compared to longer primes, while allowing a competitive memory footprint. Concerning memory size, the overall memory cost of the complete implementation is dominated by the `AND` tables. For example, by choosing $p = 53$, with the implementation described in Sect. 3, *Proposition 1* leads to an implementation of 5,014,144 bits (626.76 kilobytes), coherently with Table 1. The execution time is constant and is not dependent of the size of the prime number: $15\,744 \times 0.8 + 30\,520 \times 0.5 = 27\,855$ ns.

We remark that our new encoding allows a more efficient white-box design than the Chow *et al.* [20] one, and we also argue that our design may be adapted to a masked implementation with reduced size impact compared to the one of Biryukov *et al.* [9].

6 Further Work

In this section we provide a few ideas to further develop our encodings. We organized such ideas in two main sections. The first section suggests improvements to the side-channel security of white-boxes. Afterwards we present ideas to thwart fault attacks.

6.1 Against Side-Channel Attacks

White-box implementations are vulnerable to attacks exploiting software execution traces containing information about the memory addresses being accessed or about manipulated data. In order to complexify such attacks, one may add countermeasures. For example, one can:

– Shuffle and randomize the computations by introducing dummy operations as suggested in [10]. This can be achieved for example by computing $\mathsf{Enc}(0) \times \mathsf{Enc}(b)$ for some b, at random time.

[2] Bringer *et al.* did not provide speed figures. We used the count of monomials in Table 1 of their work and accounted one operation per monomial.

[3] Seker *et al.* did not provide memory figures. In order to obtain the memory consumption of their design we used their $(2, 1)$-masking, assumed that each gate is encoded separately (in order to avoid loops) and that each gate is encoded in 1 byte. This allows a fair comparison against for example the circuit of Biryukov *et al.* [9], where the ratio between the number of gates and the resulting size is about 6.4.

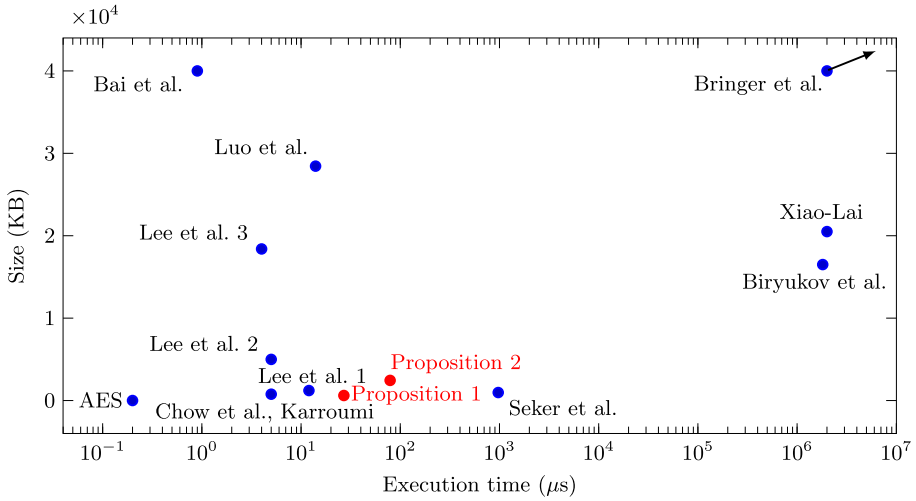

Fig. 2. Memory performances and estimated execution time of published AES-128 white-boxes.

– It is possible to mask the AES circuit prior to encoding it into tables. The impact on the size and speed would be balanced by the augmented security. We also remark that such countermeasure can easily thwart the Legendre symbol attack presented in Sect. 4.2.

6.2 Against Fault Attacks

White-box implementations are particularly vulnerable to fault attacks. An attacker can easily change the execution flow of the implementation or substitute the value of a variable [11, Sect. 7.2]. Hence, the design of a white-box must integrate countermeasures against such attacks. Typically redundancy (use of redundant representations such as a residue number system – RNS [4], or use of redundant information), error detecting or correcting techniques, or infective countermeasures [5] are used to thwart such attacks. For example, in order to introduce redundancy, one can observe that the exponents used in the encodings are values modulo r. Thus by using a composite r, it is possible to perform smaller computations modulo each prime dividing r. The result can then be recomputed by using the Chinese remainder theorem. Each of the two submodules can be used to:

– Encode the same sensitive value, which provides redundancy.
– Encode different bits of the plaintext, which provides efficiency.
– Encode the correct value on one submodule, a random value on the second one, which provides randomization.

7 Conclusion

This work addresses the problem of encoding data when building a white-box implementation. We suggest a new encoding scheme based on the Benaloh cryptosystem that allows both compactness and speed. In particular, the semi-homomorphic property of our encoding allows to drop half of the tables (in our example those used for the `XOR` operations) and to speed up computations by exchanging part of the tables (used for example in [20]) with homomorphic operations. We modify the Benaloh scheme in order to obtain an encoding that inherits the semi-homomorphic properties of the original design, while fitting the size constraints of the white-box context. Our new proposition allows the white-box designer to tune the performances and adapt the security of the implementation to meet its requirements.

As future work, we remark that it seems possible to enhance speed and memory consumption of our proposal. In this regard, a promising line of research is to parallelize multiplications and table accesses. Another direction for further work is the study of other homomorphic encryption schemes (e.g. lattice based). In particular, the study of fully homomorphic schemes may turn out advantageous. Indeed, we have shown in this paper that one can modify a semi-homomorphic scheme and use it as encoding. Thus it would be interesting to investigate the modification of a fully homomorphic scheme in a similar way. It could enable to enhance the security and the memory consumption by only keeping the input and output tables.

References

1. CHES 2017 capture the flag challenge - the WhibOx Contest - an ECRYPT white-box cryptography competition (2017). https://whibox-contest.github.io/2017/
2. CHES 2019 capture the flag challenge - the WhibOx contest edition 2 (2019). https://whibox-contest.github.io/2019/
3. Bai, K., Wu, C., Zhang, Z.: Protect white-box AES to resist table composition attacks. IET Inf. Secur. **12**(4), 305–313 (2018)
4. Bajard, J., Eynard, J., Merkiche, N.: Multi-fault attack detection for RNS cryptographic architecture. In: 23nd IEEE Symposium on Computer Arithmetic, ARITH, pp. 16–23 (2016)
5. Barbu, G., et al.: A high-order infective countermeasure framework. In: Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC) (2021)
6. Benaloh, J.: Dense probabilistic encryption. In: Selected Areas of Cryptography (1994)
7. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a white-box AES implementation. In: International Workshop on Selected Areas in Cryptography, pp. 227–240 (2004)
8. Biryukov, A., Dinu, D., Le Corre, Y., Udovenko, A.: Optimal first-order boolean masking for embedded IoT devices. In: Eisenbarth, T., Teglia, Y. (eds.) CARDIS 2017. LNCS, vol. 10728, pp. 22–41. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75208-2_2

9. Biryukov, A., Udovenko, A.: Attacks and countermeasures for white-box designs. In: International Conference on the Theory and Application of Cryptology and Information Security, pp. 373–402 (2018)
10. Biryukov, A., Udovenko, A.: Dummy shuffling against algebraic attacks in white-box implementations. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12697, pp. 219–248. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_8
11. Bock, E., et al.: White-box cryptography: don't forget about grey-box attacks. J. Cryptol. **32**, 1095–1143 (2019)
12. Bock, E.A., Amadori, A., Brzuska, C., Michiels, W.: On the security goals of white-box cryptography. IACR Trans. CHES 327–357 (2020)
13. Alpirez Bock, E., Brzuska, C., Michiels, W., Treff, A.: On the ineffectiveness of internal encodings - revisiting the DCA attack on white-box cryptography. In: Preneel, B., Vercauteren, F. (eds.) ACNS 2018. LNCS, vol. 10892, pp. 103–120. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93387-0_6
14. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: hiding your white-box designs is not enough. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 215–236. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_11
15. Boyar, J., Peralta, R.: A small depth-16 circuit for the AES S-Box. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IAICT, vol. 376, pp. 287–298. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30436-1_24
16. Bringer, J., Chabanne, H., Dottax, E.: White box cryptography: another attempt. IACR Cryptology ePrint Archive (2006)
17. Bringer, J., Chabanne, H., Le, T.H.: Protecting AES against side-channel analysis using wire-tap codes. J. Cryptogr. Eng. **2**, 129–141 (2012)
18. Calik, C.: CMT: circuit minimization team (2020). https://www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html
19. Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: ACM Workshop on Digital Rights Management, pp. 1–15 (2002)
20. Chow, S., Eisen, P., Johnson, H., Van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 250–270. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36492-7_17
21. Cox, M., Engelschall, R., Henson, S., Laurie, B., et al.: The OpenSSL Project (2002)
22. De Mulder, Y., Roelse, P., Preneel, B.: Cryptanalysis of the Xiao-Lai white-box AES implementation. In: Knudsen, L.R., Wu, H. (eds.) SAC 2012. LNCS, vol. 7707, pp. 34–49. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35999-6_3
23. De Mulder, Y., Wyseur, B., Preneel, B.: Cryptanalysis of a perturbated white-box AES implementation. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 292–310. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17401-8_21
24. Delerablée, C., Lepoint, T., Paillier, P., Rivain, M.: White-box security notions for symmetric encryption schemes. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 247–264. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43414-7_13
25. Fousse, L., Lafourcade, P., Alnuaimi, M.: Benaloh's dense probabilistic encryption revisited (2011). https://arxiv.org/pdf/1008.2991.pdf

26. Goldwasser, S., Micali, S.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, pp. 365–377 (1982)
27. Goubin, L., Masereel, J.M., Quisquater, M.: Cryptanalysis of white box DES implementations. In: International Workshop on Selected Areas in Cryptography, pp. 278–295 (2007)
28. Goubin, L., Rivain, M., Wang, J.: Defeating state-of-the-art white-box countermeasures with advanced gray-box attacks. IACR Trans. CHES **2020**(3), 454–482 (2020)
29. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_27
30. Karroumi, M.: Protecting white-box AES with dual ciphers. In: Rhee, K.-H., Nyang, D.H. (eds.) ICISC 2010. LNCS, vol. 6829, pp. 278–291. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24209-0_19
31. Kilian, J.: Founding cryptography on oblivious transfer. In: Proceedings of the Twentieth annual ACM Symposium on Theory of Computing, pp. 20–31 (1988)
32. Lee, S., Choi, D., Choi, Y.J.: Conditional re-encoding method for cryptanalysis-resistant white-box AES. ETRI J. **37**(5), 1012–1022 (2015)
33. Lee, S., Kim, M.: Improvement on a masked white-box cryptographic implementation. Cryptology ePrint Archive, Report 2020/199 (2020)
34. Lee, S., Kim, T., Kang, Y.: A masked white-box cryptographic implementation for protecting against differential computation analysis. IEEE Trans. Inf. Forensics Secur. **13**(10), 2602–2615 (2018)
35. Lepoint, T., Rivain, M., De Mulder, Y., Roelse, P., Preneel, B.: Two attacks on a white-box AES implementation. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 265–285. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43414-7_14
36. Link, H.E., Neumann, W.D.: Clarifying obfuscation: improving the security of white-box DES. In: International Conference on Information Technology: Coding and Computing (ITCC 2005)-Volume II, vol. 1, pp. 679–684. IEEE (2005)
37. Luo, R., Lai, X., You, R.: A new attempt of white-box AES implementation. In: Proceedings of 2014 IEEE International Conference on Security, Pattern Analysis, and Cybernetics (SPAC), pp. 423–429. IEEE (2014)
38. Menezes, A.J., Katz, J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography (1996)
39. Michiels, W., Gorissen, P., Hollmann, H.D.L.: Cryptanalysis of a generic class of white-box implementations. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 414–428. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04159-4_27
40. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: International Workshop on CHES, pp. 413–427 (2010)
41. Rivain, M., Wang, J.: Analysis and improvement of differential computation attacks against internally-encoded white-box implementations. IACR Trans. CHES **2019**(2), 225–255 (2019)
42. Sanfelix, E., Mune, C., de Haas, J.: Unboxing the white-box. In: Black Hat EU 2015 (2015)
43. Sasdrich, P., Moradi, A., Güneysu, T.: White-box cryptography in the gray box. In: Peyrin, T. (ed.) FSE 2016. LNCS, vol. 9783, pp. 185–203. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-52993-5_10

44. Saxena, A., Wyseur, B., Preneel, B.: Towards security notions for white-box cryptography. In: Samarati, P., Yung, M., Martinelli, F., Ardagna, C.A. (eds.) ISC 2009. LNCS, vol. 5735, pp. 49–58. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04474-8_4
45. Seker, O., Eisenbarth, T., Liskiewicz, M.: A white-box masking scheme resisting computational and algebraic attacks. Cryptology ePrint Archive, Report 2020/443 (2020)
46. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
47. Wyseur, B., Michiels, W., Gorissen, P., Preneel, B.: Cryptanalysis of white-box DES implementations with arbitrary external encodings. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 264–277. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77360-3_17
48. Xiao, Y., Lai, X.: A secure implementation of white-box AES. In: 2nd International Conference on Computer Science and its Applications, pp. 1–6. IEEE (2009)