Chapter 12
White-Box Cryptography: A Time-Security Trade-Off for the SPNbox Family

Federico Cioschi, Nicolò Fornari, and Andrea Visconti

12.1 Introduction

Traditionally, cryptographic primitives are designed to protect data and keys in the black-box attack model, in which the communication end points are trusted, meaning that the cipher execution (encryption/decryption, instantiation with a secret key) cannot be observed or tampered with. However, the assumptions made in the past may often not be applicable in the current technology, such as DRM applications, Pay Tv boxes, and smartphones. For this reason, we refer to the *white-box model* as an attack model in which the adversary has total visibility of the software implementation of the cryptosystem, and full control over its execution platform.

White-Box Cryptography was originally defined [1] as an obfuscation technique intended to implement cryptographic primitives in such a way that an adversary having full access to the implementation and execution platform is unable to extract

Some of this work was done as part of the author Federico Cioschi's BSc thesis, Department of Computer Science, Università degli Studi di Milano.

Some of this work was done as part of the author Nicolò Fornari's MSc thesis, Department of Mathematics, University of Trento.

F. Cioschi · A. Visconti (✉)
Department of Computer Science, Università degli Studi di Milano, Milano, Italy
e-mail: federico.cioschi@studenti.unimi.it; andrea.visconti@unimi.it

N. Fornari
The Akkademy, Geneva, Switzerland
e-mail: nicolo.fornari@akka.eu

I. Woungang, S. K. Dhurandher (eds.), *2nd International Conference on Wireless Intelligent and Distributed Environment for Communication*, Lecture Notes on Data Engineering and Communications Technologies 27,
https://doi.org/10.1007/978-3-030-11437-4_12

key information. One may wonder why the adversary should be interested in recovering the key when he/she controls the encryption/decryption software and can use it to encrypt/decrypt data. The reader should be aware that such software might be subject to restrictions as in the case of DRM applications. If a malicious user is able to recover the secret key, then she/he can encrypt or decrypt the data with *any* software on *any* host. In the case of applications enforcing DRM schemes (for example, Sky Go [2], Netflix [3], and Spotify [4]), a key recovery attack would allow an adversary to illegally distribute content to non-subscribers of the service offered by the application. Notice that the definition of white-box cryptography as in [1] is limited to key recovery attempts and does not take into account other attacks such as code lifting, where the attacker attempts to isolate the program code from the implementation environment and directly uses the code itself as a larger key. Therefore, we refer to [5] according to which a white-box implementation of cryptographic primitives does not present any advantage for a computationally bounded adversary in comparison to the adversary dealing with the implementation as a black-box.

White-Box implementations of DES and AES were first proposed by Chow et al. in [6] and [1]. Their approach was to find a representation of the cryptographic algorithm as a network of look-ups in randomized and key-dependent tables. These papers, as well as some others [7, 8], are subjected to algebraic attacks [9–12]. However, such attacks require knowledge of the internal data representation used by the implementation, meaning in practice significant reverse engineering efforts.

A breakthrough from the attacker's side came with the work of Bos et al. [13] who proposed two new attack paradigms which can be automated and do not require reverse engineering efforts. The first attack is known as differential fault analysis (DFA) and can be regarded as the software counterpart of fault-injection attacks on cryptographic hardware. The second one, known as differential computational analysis (DCA), can be thought as a side-channel attack adapted to the white-box attack model.

The DCA attack collects software execution traces for several plaintext encryptions and uses the collected data to perform an analysis similar to the well-known differential power analysis (DPA) to recover the secret key. Since the software traces contain time demarcated physical addresses of memory locations being read and written to, they leak the values of the inputs to the various look-up tables accessed during the white-box encryption operation, which leak enough information to perform the power attack. In [14], Banik et al. further investigate the DCA attack proposing software countermeasures such as randomization of the locations of the look-up tables in memory, in addition to control flow obfuscation. They also develop an attack based on software traces called zero difference enumeration (ZDE). The attack records software traces for several pairs of strategically selected plaintexts and performs a statistical test on the difference of the traces to extract the secret key.

Whereas all published white-box implementations for standard cryptographic algorithms such as DES and AES are prone to practical key extraction attacks as just described, there have been two dedicated design approaches for white-box block ciphers: ASASA by Birykov et al. [15] and SPACE by Bogdanov and Isobe [16]. While ASASA suffers from decomposition attacks, SPACE presents a design for which security against key extraction in the white-box context reduces to the well-studied problem of key recovery for block ciphers in the standard black-box setting. However, SPACE imposes a sometimes prohibitive performance overhead in the real world as it needs many AES calls to encrypt a single block. In [17], Bogdanov et al. address the issue by designing a family of dedicated white-box block ciphers SPNbox and a family of underlying small block ciphers with software efficiency and constant-time execution in mind.

In this chapter, we modify the underlying small block ciphers to increase the number of bits of the key used in each round. This approach can be exploited to make the algorithm faster (we are using the same number of bits of the key reducing the number of rounds) or more secure (we are using the same number rounds increasing the number of bits of the key) than previous.

The remainder of the chapter is organized as follows. In Sect. 12.2, we briefly introduce the white-box implementations of well-known algorithms such as AES. In Sects. 12.3 and 12.4, we describe two families of block ciphers, i.e., the SPACE family and the SPNbox family, designed to be white-box friendly. In Sect. 12.5, we (a) suggest how to speed up the SPNbox family and (b) evaluate the performance of the suggested approach. Finally, discussion and conclusions are drawn in Sect. 12.6.

12.2 White-Box Constructions and Attacks

The idea to avoid key recovery attacks is to mathematically fuse the key with the encryption routine, formally, given a block cipher ϕ, one wants to construct $\psi : \mathbb{F}^n \to \mathbb{F}^n$ such that, fixed a key $\bar{k} \in \mathbb{F}^l$, then $\phi(x, \bar{k}) = \psi(x) \ \ \forall x \in \mathbb{F}^n$. Clearly, the secret key k should not be easily retrievable from an attacker knowing ϕ and ψ.

Example Let ϕ, ψ be defined as:

$$\begin{aligned} \phi(x) &:= k + x \mod 4 \quad x \in \{0, \dots, 3\} \\ \psi(x) &:= S[x] \qquad\qquad S = [3, 0, 1, 2] \end{aligned}$$

With $k = 3$, we can think of ψ as a trivial white-box implementation of ϕ. In terms of implementation, we can think of ψ as a look-up table.

The first white-box construction for AES was proposed by Chow et al. in [1]. Their work came from a simple idea: given a fixed encryption key and a block cipher, i.e., AES-128, building a look-up table mapping all the possible plaintext to the respective ciphertext is secure against key extraction. Of course for a mapping $\mathbb{F}^{128} \to \mathbb{F}^{128}$ such construction is utterly unfeasible if we use only a huge table. However, many small tables could be used instead.

The approach taken by Chow et al. was to represent a block cipher ϕ as a network of key-dependent look-up tables (see Fig. 12.1 and [1] for details).

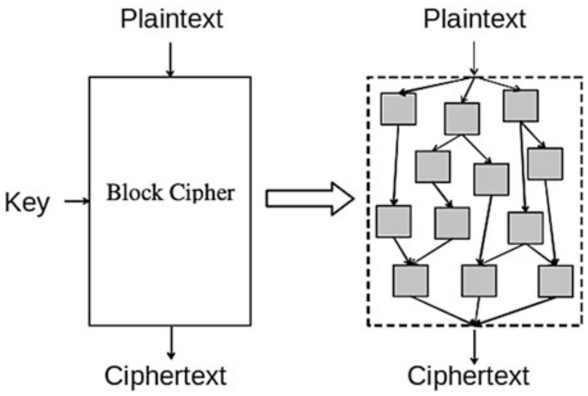

Fig. 12.1 Table-based white-box implementation: the key *k* is scrambled by a network of look-up tables

Table 12.1 To the best of our knowledge for each white-box AES implementation available in the literature, there exists an efficient attack

White-box AES impl.	Cryptanalysis	Work factor
Chow [1]	[9]	2^{30}
Karroumi [7]	[10, 12]	2^{22}
Xiao Lai [8]	[10]	2^{32}
Xiao Lai [8] generic linear version	[10]	2^{38}
Xiao Lai [8] affine/non-affine version	[11]	At least 2^{49}

Chow's pioneering work became a reference point for some of the subsequent proposals such as [7] and [8]. For each construction however, an attack was published as summarized in Table 12.1.

Interestingly, the attacks in Table 12.1 require knowledge of the internal data representation used by the implementation. In practice, the level of implementation knowledge required by the attacker is only attainable through significant reverse engineering efforts.

A significant breakthrough from the attacker's perspective became possible by shifting the focus from pure algebraic attacks, as in Table 12.1, to side channel attacks. In [13], Bos et al. describe new approaches to assess the security of white-box implementations. These approaches require *neither* knowledge about the look-up tables used *nor* expensive reverse engineering effort.

Bos et al. introduce a *differential fault analysis* (DFA) attack which is the software counterpart of fault-injection attacks on cryptographic hardware (for more details regarding fault-injection attacks, see [18]). They also describe a *differential computational analysis* (DCA) attack, which is the software counterpart of the differential power analysis for cryptographic hardware (for more details regarding DPA attacks, see [19]).

There are two fundamental concepts that make DFA and DCA attacks so effective:

- Chow's construction is a white-box implementation of AES and, despite being based on look-up tables, the regular structure of AES (SubBytes, Shiftrows, MixColumns, and AddRoundKey) is still preserved.
- Traditional side channel attacks such as fault injection and differential power analysis are extremely more effective in the white-box attack model since the attacker has full control over the execution platform and can thus perform direct measures.

Considering the effectiveness of DCA and DFA attacks, it is time to wonder whether AES is a suitable cipher in the white-box attack model and to consider alternative approaches. Instead of trying to modify well-known ciphers, designed in the black-box model, in order to be resistant in the white-box attack model, it is worth trying to design new ciphers with the white-box attack model in mind. An example of such cipher is SPACE, recalled in Sect. 12.3.

12.3 SPACE

SPACE is a block cipher proposed by Bogdanov and Isobe in [16]. This cipher presents an interesting design for which security against key extraction in the white-box context reduces to the well-studied problem of key recovery for block ciphers in the standard black-box setting.

12.3.1 SPACE Design

SPACE is a generalized Feistel network [20] which encrypts a message $m \in \mathbb{F}^n$ with a secret key $k \in \mathcal{K}$ to a ciphertext $c \in \mathbb{F}^n$. There are three positive integers that we will often use in the description of SPACE: n, n_a, and n_b. In [16], Bogdanov and Isobe fix $n = 128$, $n_a \in \{8, 16, 24, 32\}$, and $n_b = n - n_a$.

The encryption works as follows:

1. Let X^r be the state at round r. The state is seen as l vectors $x_i^r \in \mathbb{F}^{n_a}$:

$$X^r = \{x_0^r, x_1^r, \dots, x_{l-1}^r\}$$

   where $l = n/n_a$.
2. X^0 is initialized with the value of the plaintext message m.

3. For $r \in \{1, \dots, R+1\}$, the state updates as follows:

$$X^{r+1} = \left(F^r_{n_a}(x^r_0) \oplus (x^r_1 || x^r_2 || \dots || x^r_{l-1})\right) || x^r_0$$

where $F^r_{n_a} : \mathbb{F}^{n_a} \rightarrow \mathbb{F}^{n_b}$ is the Feistel function and $||$ denotes concatenation.
4. X^{R+1} is the ciphertext c.

The encryption is simple: each round takes x^r_0 as input to the Feistel function, then adds $F^r_{n_a}(x^r_0)$ to the rest of the state $(x^r_1 || \dots || x^r_{l-1})$. The outcome of this operation is saved as the first n_b bits of the new state, while the last n_a are filled by x^r_0.

12.3.2 *Feistel Function as a Look-Up Table*

Let π_t be a projection, with $t \in \{1, \dots, 2n\}$, defined as:

$$\begin{aligned} \pi_t : \quad & \mathbb{F}^{2n} \quad \rightarrow \quad \mathbb{F}^t \\ & (x_1, \dots, x_{2n}) \mapsto (x_1, \dots, x_t) \end{aligned}$$

If we think of $x \in \mathbb{F}^{2n}$ as a vector of bits, we can use π_t to select the t *most* significant bits of x.

Definition 1 Let the Feistel function $F^r_{n_a}$ used by SPACE be defined as:

$$\begin{aligned} F^r_{n_a}(x) : \mathbb{F}^{n_a} &\rightarrow \mathbb{F}^{n_b} \\ x &\mapsto (\pi_{n_b}(\phi_k(\overbrace{0, \dots, 0}^{n_b} || x))) \oplus r \end{aligned}$$

where ϕ_k is a block cipher and r is the round number represented in binary using n_b digits, thus seen as an element of $\mathbb{F}^{n_b}$.

Let us isolate the part of the Feistel function that is round independent.

Definition 2 Let F'_{n_a} be the round independent part of F_{n_a}, defined as:

$$\begin{aligned} F'_{n_a} : \mathbb{F}^{n_a} &\rightarrow \mathbb{F}^{n_b} \\ x &\mapsto \pi_{n_b}(\phi_k(\overbrace{0, \dots, 0}^{n_b} || x)) \end{aligned}$$

Now observe that, compared to traditional Feistel networks, SPACE does not use round keys. There is one secret key k used by ϕ_k. Such secret key cannot be hardcoded, hence $\mathbb{F}'_{n_a}$ is implemented as a look-up table. The reader might wonder

Fig. 12.2 The value of each image of $F'_{n_a}(x)$ is saved as a row in a look-up table. Each row is indexed by the value of $x, x \in \{0, \dots, 2^{n_a} - 1\}$

$$\begin{aligned}
(\overbrace{0, \dots, 0}^{n_b} \,||\, \overbrace{0, \dots 0, 0}^{n_a}) &\mapsto \overbrace{\pi_{n_b}(\phi_k(0, \dots, 0||0, \dots 0, 0)}^{n_b} \\
(0, \dots, 0||0, \dots 0, 1) &\mapsto \pi_{n_b}(\phi_k(0, \dots, 0||0, \dots 0, 1)) \\
&\vdots \\
(0, \dots, 0||1, \dots 1, 1) &\mapsto \pi_{n_b}(\phi_k(0, \dots, 0||1, \dots 1, 1))
\end{aligned}$$

Table 12.2 Table size for different values of n_a and comparison with other white-box implementations

Cipher	Table size
SPACE-(8,300)	3.84 KB
SPACE-(16,128)	918 KB
SPACE-(24,128)	218 MB
SPACE-(32,128)	51.5 GB
AES (Chow et al.)	752 KB
AES (Xiao Lai)	20.5 MB

We keep the notation SPACE(n_a,R) as in [16], where R is the suggested number of rounds

the reason for designing SPACE over another block cipher ϕ_k when ϕ_k could be implemented as a look-up table directly. It happens that this second option is not possible. If we were to implement ϕ_k as a look-up table, we would need $2^n \cdot n$ bits of space:

$$\begin{aligned}
(\overbrace{0, \dots 0, 0}^{n}) &\mapsto \overbrace{\phi_k(0, \dots 0, 0)}^{n} \\
(0, \dots 0, 1) &\mapsto \phi_k(0, \dots 0, 1) \\
&\vdots \\
(1, \dots 1, 1) &\mapsto \phi_k(1, \dots 1, 1)
\end{aligned}$$

Notice that for $n = 128$ such look-up table would be infeasible to construct. The idea of Bogdanov and Isobe (see Fig. 12.2) is to truncate the output of ϕ_k, computed over a smaller domain:

Since the first n_b zeros are used as padding in order to form an n-bit input to provide to ϕ_k, there is no need to store them, hence the look-up table implementation requires $2^{n_a} \cdot n_b$ bits. In Table 12.2, we report the table size of SPACE for $n_a \in \{8, 16, 24, 32\}$.

It is self-evident that not all values of n_a are apt for a real-world application: $n_a = 32$ requires 51.5 GB of storage and even $n_a = 24$, requiring 218 MB, is not suitable for all applications. However, $n_a = 16$ requires only 918 KB, placing itself at the same level of Chow [1] as storage size, see Table 12.2.

12.4 The SPNbox Family

As shown in Sect. 12.3, the SPACE family of space-hard block ciphers [16], using a Feistel structure, offers interesting security properties but it prevents the exploitation of parallel execution. However, as described in [17], using an SPN-type design it is possible to satisfy the requirement of parallelism maintaining a sufficiently high level of space hardness. Therefore, in 2016 Bogdanov et al. described the SPNbox [17] family of space-hard block ciphers, whose structure is shown in Fig. 12.3. Let us explain this approach more formally.

SPNbox-n_{in} is a substitution–permutation network (SPN) with a block length of n bits, a k-bit secret key, and based on n_{in}-bit substitution boxes.

State The state of SPNbox-n_{in} can be represented as a vector of $t = n/n_{in}$ elements of n_{in} bits each:

$$X = \{X_0, \ldots, X_{t-1}\}$$

Key Schedule The k-bit master key is expanded—i.e., $k_0, \ldots, k_{R_{n_{in}}}$ round keys of n_{in} bits—using a key derivation function (KDF)[1]:

$$(k_0, \ldots, k_{R_{n_{in}}}) = KDF(k, n_{in} \cdot (R_{n_{in}} + 1))$$

Round Transformation We encrypt a plaintext X^0 and we get a ciphertext X^R, by applying the following R transformations—e.g., $R = 10$:

$$X^R = (\bigcirc_{r=1}^{R} (\sigma^r \circ \theta \circ \gamma))(X^0)$$

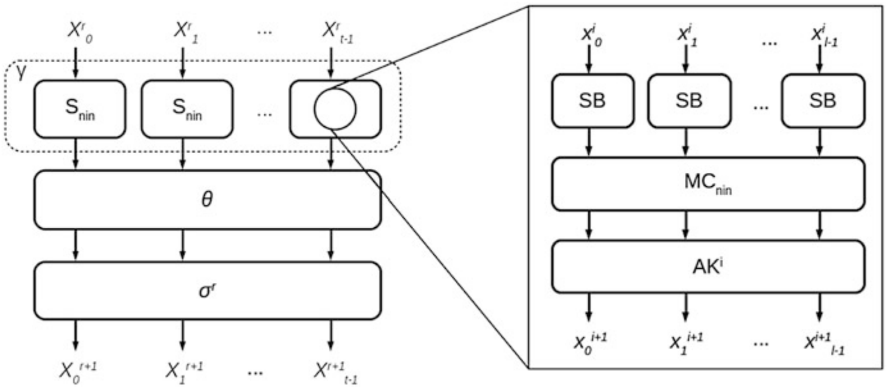

Fig. 12.3 The SPNbox structure with a zoomed view on the inner round

[1]For example, PBKDF2 [21–23], ARGON2 [24], Scrypt [25], and so on.

The nonlinear layer γ is a substitution layer in which t key-dependent identical bijective n_{in}-bit S-boxes are applied to the state:

$$\gamma : \mathbb{F}(2^{n_{in}})^t \to \mathbb{F}(2^{n_{in}})^t$$
$$(X_0, \ldots, X_{t-1}) \mapsto (S_{n_{in}}(X_0), \ldots, S_{n_{in}}(X_{t-1}))$$

These identical S-boxes are realized by an internal small block cipher of block length n_{in} bit.

The linear layer θ, a diffusion layer, applies a $t \times t$ MDS matrix to the state:

$$\theta : \mathbb{F}(2^{n_{in}})^t \to \mathbb{F}(2^{n_{in}})^t$$
$$(X_0, \ldots, X_{t-1}) \mapsto (X_0, \ldots, X_{t-1}) \cdot M_{n_{in}}$$

The affine layer σ^r adds round-dependent constants to the state:

$$\sigma^r : \mathbb{F}(2^{n_{in}})^t \to \mathbb{F}(2^{n_{in}})^t$$
$$(X_0, \ldots, X_{t-1}) \mapsto (X_0 \oplus C_0^r, \ldots, X_{t-1} \oplus C_{t-1}^r),$$

with $C_i^r = (r-1) \cdot t + i + 1$ for $0 \leq i \leq t-1$.

The Underlying Small Block Ciphers The key-dependent identical n_{in}-bit S-boxes in the γ layer are block ciphers themselves. They are based on the round transformation of AES and consist of $R_{n_{in}}$ rounds operating on a state $x = \{x_0, \ldots, x_{l-1}\}$ of l bytes, where $l = n_{in}/8$ [2]:

$$S_{n_{in}} : \mathbb{F}(2^8)^l \to \mathbb{F}(2^8)^l$$
$$x \mapsto \left(\bigcirc_{i=1}^{R_{n_{in}}} (AK^i \circ MC_{n_{in}} \circ SB)\right)(AK^0(x))$$

Notice that: (a) the number of rounds $R_{n_{in}}$ suggested in [17] is $R_{32} = 16$, $R_{24} = 20$, $R_{16} = 32$, and $R_8 = 64$; (b) different matrices are adopted in the $MC_{n_{in}}$ round transformation. More precisely, for $n_{in} = 32$ we use the MC matrix of AES, while in the other cases a sub-matrix of the original one is used. Interestingly, when $n_{in} = 8$, $MC_{n_{in}}$ represents the identity mapping. Notice that, as it happens for the Feistel function in SPACE, in the white-box setting the small block ciphers $S_{n_{in}}$ are implemented as look-up tables.

[2]SB, MC, and AK refer to the AES transformations SubBytes, MixColumns, and AddRoundKey, respectively.

12.5 Our Contribution

Three possible issues can be identified in the solution presented in Sect. 12.4.

1. In [17], the authors proposed a black-box implementation of the cipher that uses the AES-NI instructions. During the encryption phase, when $n_{in} = 32$ bits, the A_{32} matrix used by the MixColumns transformation is the same as that used in AES MixColumns. This is true also for $n_{in} = 24$ and $n_{in} = 16$. However, in these cases the A_{32} matrix is not fully used, but two sub-matrices of A_{32}—respectively called A_{24} and A_{16}—are involved in the computation. On the contrary, a different approach has to be adopted in the decryption phase. Let us suppose a scenario in which several clients have to communicate with a server. In this scenario, a black-box implementation of the decryption system may be needed (server side). Such decryption requires the inverse matrices of A_{24} and A_{16} which are not sub-matrices of the inverse of A_{32}. The problem lies on the `aesdec` instruction, provided by AES-NI for decryption, because this instruction is based only on the inverse matrix of A_{32}. In this case, it is not possible to use AES-NI instructions for $n_{in} = 24, 16$.
2. As mentioned in [17], the `aesenc`[3] instruction implies an overhead which depends on the block size (n_{in}). Since increasing parts of the instruction state are not used, it may happen that at halving of n_{in} the overhead doubles. Therefore, the number of `aesenc` instructions needed to encrypt a 128-bit SPNbox state doubles too.
3. Finally, as stated in [17], a possible drawback could be an efficiency bottleneck about the key mixing in the small internal block ciphers. Indeed, the smaller the block size, the more rounds are needed to avoid meet-in-the-middle attacks. This raises the question of how to design an internal block cipher with a faster and secure key mixing.

We try to face these issues by designing an internal block cipher with a faster and secure key mixing. In particular, we modify the inner round (shown in Fig. 12.3) and increase the number of bits of the key used in each round. In doing so, we suggest to replace the classical ShiftRow transformation, omitted in SPNbox [17], with a key-dependent circular bit shift. More precisely, if $n_{in} = 8$, then three bits of the key are required to execute a circular shift on the state (see Fig. 12.4).

This approach allows us to use 11 bits of the key in each round i: eight of them for the AK^i transformation and three for the BitShift. With a larger state, i.e., 16, 24, 32 bits, an independent circular shift is performed on each byte of the state. This means that when $n_{in} = 16, 24, 32$, the number of bits of the key used are 22, 33, and 44, respectively. Although the modified internal structure of AES precludes

[3]Provided by AES-NI to make one round of AES encryption.

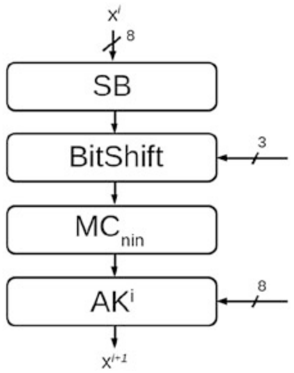

Fig. 12.4 A BitShift transformation to increase the number of bits of the key used

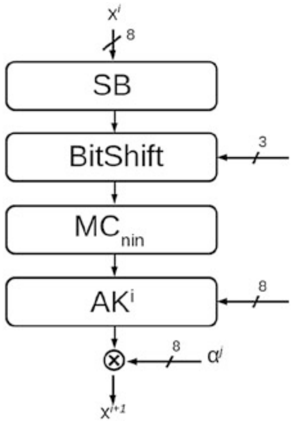

Fig. 12.5 A modular multiplication to increase the number of bits of the key used

the possibility to make use of the fast AES-NI instructions, the idea suggested can be exploited in two ways:

- If we leave the total number of bits of the key unchanged and increase the number of bits used in each round, thus we are reducing the number of inner rounds, that means speeding up the small internal block cipher;
- If we increase the total number of bits of the key used and leave the number of inner rounds unchanged, thus we are providing the possibility to engineer a block cipher with an improved level of security against meet-in-the-middle attacks.

A further possibility is to add a key-dependent polynomial multiplication at the end of each round. This approach can be implemented with, or without, the BitShift transformation (see Fig. 12.5), allowing the use of the AES-NI instructions. In both cases, the state is multiplied by α^j, where α is a primitive element of $\mathbb{F}(2^{n_{in}})$ and j is an n_{in}-bit exponent provided by the KDF.

Table 12.3 Comparison of the γ layer with and without the BitShift transformation

	γ	γ with BitShift
$n_{in} = 32$, encryption	1.178316 s	0.955048 s
$n_{in} = 32$, decryption	1.447580 s	1.168507 s
$n_{in} = 16$, encryption	3.946748 s	3.222751 s
$n_{in} = 16$, decryption	4.193261 s	3.308678 s
$n_{in} = 8$, encryption	2.547156 s	2.192452 s
$n_{in} = 8$, decryption	2.564750 s	2.250102 s

12.5.1 Performance Evaluation

We compared the performance of the internal layer γ in the black-box setting with and without BitShift transformation for different n_{in} sizes. We run our code on a laptop equipped with Ubuntu 18.04.1 LTS 64 bit, 8 GB RAM, and an Intel® Core™ i3-330M @ 2.13 GHz processor with 3 MB cache. We compile our source code with GCC 7.3.0, `-O3` optimization enabled. Table 12.3 shows the time required to encrypt/decrypt one million of different 128-bit plaintexts using the same key.

SPNbox layer γ (see Fig. 12.3) uses 512 key bits (in addition to those needed for the initial AddRoundKey AK^0)—i.e., 512 bit = 16 round $\times$ 32 bit ($n_{in} = 32$), or 512 bit = 32 round $\times$ 16 bit ($n_{in} = 16$), or 512 bit = 64 round $\times$ 8 bit ($n_{in} = 8$). Setting to 512 bit the minimum amount of key bits to be used, our solution will execute: 12 rounds, using 528 key bits ($n_{in} = 32$); 24 rounds, using 528 key bits ($n_{in} = 16$); and finally 47 rounds, using 517 key bits ($n_{in} = 8$).[4]

12.6 Conclusions and Future Works

White-box cryptography aims to ensure the security of cryptographic algorithms in an untrusted environment where an adversary has total visibility of the cryptographic implementations and full control over the software execution platform. In order to make well-known ciphers safe in a white-box context, researchers suggested a number of solutions [1, 6–8]. However, such implementations are subjected to algebraic attacks and side channel attacks, thus researchers developed new ciphers—e.g., SPACE [16] and the SPNbox family [17]—with the white-box attack model in mind.

In this context, our aim is to improve the approach adopted in the SPNbox family [17], focusing on the internal small block ciphers used by Bogdanov et al. In particular, we suggest to increase the number of bits of the key used in each round, replacing the classical ShiftRow operation with a key-dependent circular bit shift,

[4]Notice that, when $n_{in} = 32, 16, 8$, the number of key bits used has to be incremented by the appropriate number of key bits used by AK^0.

or introducing a key-dependent polynomial multiplication over the field $\mathbb{F}(2^{n_{in}})$ at the end of the round. The approach suggested can be exploited in two ways: to make the algorithm faster than previous approach, or to make the algorithm more secure against the meet-in-the-middle attack. The testing activities executed on a consumer laptop showed a reduction between 12.27% and 21.10% of the execution time (without AES-NI instructions) of layer γ. Our future work will focus on further increasing the number of bits used by the inner round.

References

1. S. Chow, P. Eisen, H. Johnson, P.C. Van Oorschot, White-box cryptography and an AES implementation, in: International Workshop on Selected Areas in Cryptography (Springer, Berlin, 2002), pp. 250–270
2. Sky Go, http://go.sky.com/. Accessed 13 Nov 2018
3. Netflix, https://www.netflix.com. Accessed 13 Nov 2018
4. Spotify, https://www.spotify.com/. Accessed 13 Nov 2018
5. B. Wyseur, White-Box Cryptography. Ph.D. Thesis, KU Leuven, Department of Mathematics (2009)
6. S. Chow, P. Eisen, H. Johnson, P.C. Van Oorschot, A white-box DES implementation for DRM applications, in ACM Workshop on Digital Rights Management (Springer, Berlin, 2002), pp. 1–15
7. M. Karroumi, Protecting white-box AES with dual ciphers, in International Conference on Information Security and Cryptology (Springer, Berlin, 2010), pp. 278–291
8. Y. Xiao, X. Lai, A secure implementation of white-box AES, in 2nd International Conference on Computer Science and its Applications, 2009, CSA'09 (IEEE, Piscataway, 2009), pp. 1–6
9. O. Billet, H. Gilbert, C. Ech-Chatbi, Cryptanalysis of a white box AES implementation, in International Workshop on Selected Areas in Cryptography (Springer, Berlin, 2004), pp. 227–240
10. Y. De Mulder, P. Roelse, B. Preneel, Cryptanalysis of the Xiao–Lai white-box AES implementation, in International Conference on Selected Areas in Cryptography (Springer, Berlin, 2012), pp. 34–49
11. W. Michiels, P. Gorissen, H.D. Hollmann, Cryptanalysis of a generic class of white-box implementations, in International Workshop on Selected Areas in Cryptography (Springer, Berlin, 2008), pp. 414–428
12. T. Lepoint, M. Rivain, Y. De Mulder, P. Roelse, B. Preneel, Two attacks on a white-box AES implementation, in International Conference on Selected Areas in Cryptography (Springer, Berlin, 2013), pp. 265–285
13. E.A. Bock, J.W. Bos, C. Brzuska, C. Hubain, W. Michiels, C. Mune, E.S. Gonzalez, P. Teuwen, A. Treff, White-box cryptography: don't forget about grey box attacks. Cryptology ePrint Archive, Report 2017/355 (2017)
14. S. Banik, A. Bogdanov, T. Isobe, M. Jepsen, Analysis of software countermeasures for whitebox encryption. IACR Trans. Symmetric Cryptol. **2017**(1), 307–328 (2017)
15. A. Biryukov, C. Bouillaguet, D. Khovratovich, Cryptographic schemes based on the ASASA structure: black-box, white-box, and public-key (extended abstract), in P. Sarkar, T. Iwata (eds.) Advances in Cryptology – ASIACRYPT 2014 (Springer, Berlin, 2014), pp. 63–84
16. A. Bogdanov, T. Isobe, White-box cryptography revisited: space-hard ciphers, in Proceedings of the 22nd ACM SIGSAC conference on computer and communications security (ACM, New York, 2015), pp. 1058–1069

17. A. Bogdanov, T. Isobe, E. Tischhauser, Towards practical whitebox cryptography: optimizing efficiency and space hardness, in International Conference on the Theory and Application of Cryptology and Information Security (Springer, Berlin, 2016), pp. 126–158
18. P. Dusart, G. Letourneux, O. Vivolo, Differential fault analysis on AES, in International Conference on Applied Cryptography and Network Security (Springer, Berlin, 2003), pp. 293–306
19. P. Kocher, J. Jaffe, B. Jun, P. Rohatgi, Introduction to differential power analysis. J. Cryptogr. Eng. **1**(1), 5–27 (2011)
20. H. Feistel, Cryptography and computer privacy. Sci. Am. **228**(5), 15–23 (1973)
21. K. Moriarty, B. Kaliski, A. Rusch, PKCS# 5: Password-Based Cryptography Specification Version 2.1. RFC 8018 (2017)
22. A. Visconti, S. Bossi, H. Ragab, A. Calò, On the weaknesses of PBKDF2, in ed. by M. Reiter, D. Naccache. Cryptology and Network Security (Springer, Berlin, 2015), pp. 119–126
23. A. Visconti, F. Gorla, Exploiting an HMAC-SHA-1 optimization to speed up PBKDF2. IEEE Trans. Dependable Secure Comput. (2018). https://doi.org/10.1109/TDSC.2018.2878697
24. A. Biryukov, D. Dinu, D. Khovratovich, Argon2 (version 1.2). https://password-hashing.net/submissions/specs/Argon-v3.pdf. Accessed 13 Nov 2018
25. C. Percival, S. Josefsson, The scrypt Password-Based Key Derivation Function. RFC 7914 (2016)