SCIENCE CHINA
Information Sciences

• **Supplementary File** •

Protecting White-Box Cryptographic Implementations with Obfuscated Round Boundaries

Tao XU[1,2,3], Chuankun WU[4*], Feng LIU[1,2,5*] & Ruoxin ZHAO[1,2]

[1]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093*, China;*
[2]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing* 100049*, China;*
[3]*New United Group Co., Ltd., Changzhou, Jiangsu* 213166*, China;*
[4]*School of Mathematics, Shandong University, Jinan* 250100*, China;*
[5]*School of Big Data and Computer Science, Guizhou Normal University, Guiyang, Guizhou* 550001*, China*

Appendix A Discussions on Published White-Box Cryptographic Implementations

Appendix A.1 Security Goals

By now, there are not standard specifications for white-box cryptography. So the implementations may vary. The prime security goal of a WBC implementation is unbreakability. Yet, some other goals are needed in real applications. The goals are described as follows. Each published WBC implementation satisfies a subset of them.

- **Unbreakability**. An adversary cannot extract the secret key of the WBC implementation.
- **One-Wayness**. The WBC implementation enables one to encrypt(or decrypt) at will. But it should not allow the reverse process.
- **Incompressibility**. It should be hard for an adversary to compress the WBC implementation into an equivalent beyond a certain prescribed size.
- **Traceability**. The WBC implementation could have several versions of the same functionality. Different versions are assigned to different users. If a malicious user shares his own, the system administrator can trace him by identifying his abused version. Traceability is always achieved from the random components which are canceled internally and have no effect on the final output results.

Appendix A.2 Protocol-Assisted Incompressibility

Biryukov et al.'s **ASASA** structure is realized as a single lookup table [8]. They introduced 'weak white-box security' to describe the security goal against decomposing this table.

Definition 1 (Weak White-Box Security)**.** Let O_{E_K} be a white-box implementation for E_K, where E is an encryption algorithm and K is the key. O_{E_K} is a T-secure weak white-box implementation if it is computationally hard to obtain an equivalent key of length less than T given full access to O_{E_K}.

It means that it should be computationally hard for an adversary to find out any compact equivalent representation of the WBC implementation, i.e., **incompressibility**. However, in [9], the **ASASA** lookup table was decomposed by Dinur et al. successfully. They found small equivalents of the secret affine (A) and nonlinear (S) layers. An adversary could distribute the equivalents to illegal users.

If Biryukov et al.'s WBC implementation follows the goal of **traceability**, it can still maintain **incompressibility** in a sense with the assistance of a challenge-response protocol, though there exists a decomposing attack. For example, in a digital rights management (DRM) scheme, the content server regularly queries the hash value of a randomly selected segment of the WBC implementation running in a client. If the answer from a client is incorrect, the server will abort the digital content service. It forces the client to give up using a compressed illegal version from an adversary. We call this kind of **incompressibility** in reality is **protocol-assisted incompressibility**.

* Corresponding author (email: 13520965063@163.com, liufeng@iie.ac.cn)

Definition 2 (Protocol-Assisted Incompressibility)**.** Let O_{D_K} be a white-box implementation for D_K, where D is a decryption algorithm and K is the key. Let P be a tracing protocol. Server S can provide encrypted contents to client C only when P is met. Given full access to O_{D_K} running in C, O_{D_K} is a protocol-assisted incompressible white-box implementation if it is hard to obtain a smaller equivalent of O_{D_K} to meet P at runtime.

Unfortunately, Biryukov et al.'s scheme does not follow **traceability**. Then it is not of protocol-assisted incompressibility.

Appendix A.3 Mitigating Code Lifting

Code lifting means that an adversary can lift the whole WBC implementation as an equivalent large key to copy the functionality. Following the strict definition of white-box attack model, it is infeasible to prevent code lifting facing a mighty adversary. However, most commonly in reality, the adversaries are not so mighty and the illegal users, who get an copy of the WBC implementation, are not so mighty. There are two practical methods to mitigate code lifting attacks: **(i)** external encodings; **(ii)** large size of implementation which is beyond an attacker's processing capacity to copy and distribute the software's functionality.

The second method is relevant to the concept of memory hardness and can be evaluated by weak white-box security and **incompressibility** [8]. A white-box implementation is memory-hard if it requires a pre-specified and large enough amount of memory. In [10], Bogdanov et al. generalized 'weak white-box security' to 'space hardness' which assesses the difficulty of code lifting by the amount of the data.

Definition 3 ((M, Z)-space hardness [10])**.** An implementation of a block cipher E_K is (M, Z)-space hard if it is infeasible to encrypt (decrypt) any randomly drawn plaintext (ciphertext) with probability of more than 2^{-Z} given any code (table) of size less than M.

There are several useful features of using a white-box implementation of high level space hardness. First, an attacker is unable to reduce the implementation size and have to share the entire implementation to unauthorized users. The act of illegal distribution will be limited. Second, if a large proportion of users sacrifice an affordable amount of memory to white-box implementations, it will be very difficult or even impossible for some government sectors to deal with the keys of all users simultaneously. Applying mass and untargeted surveillance, just as Snowden described in his revelations, will be infeasible. Finally, because it is difficult to send the whole secret material out through the networks, the damage of having malware in a top secret network system will be relieved.

Appendix A.4 Characteristics of White-Box Cryptography

Through observing published implementations, we summarize the characteristics of white-box cryptography as follows.

- **Practical Security:** White-box cryptography is a technique to provide practical security for software applications in open devices. Though there still lacks solid theoretic foundations, it has been used in many actual industrial applications, e.g. Irdeto's DRM solution, Bell ID's HCE solution and whiteCryption's Cryptanium Secure Key Box.
- **Security vs Efficiency:** The general way to hide the secret key is to use key-dependent lookup tables. It makes the application more secure, but meanwhile, it causes more complexity and cost. The designer should keep the balance between security and efficiency. However, from another point of view, the large size of white-box implementations could even be a security measure (space hardness).
- **Server Load:** In a white-box solution, the server is still considered trustworthy. So it is not necessary to use white-box implementations of cryptographic algorithms on the server side. Then in the situation of serving many users, the total memory size will not be a burden for the server.

Appendix B White-Box AES with Obfuscated Round Boundaries

We take AES for example to illustrate the details of our method. Furthermore, a discussion of the security and the running efficiency is given.

Appendix B.1 Some Notations

- Let $x \oplus y$ denote the bit-wise exclusive-or (XOR) of x and y, where x and y are bit-strings of equal length. $\oplus_c$ means the function $f(x) = x \oplus c$. Here, c is a constant.
- Let $f \circ g$ denote the composition of two functions f and g, i.e., $f \circ g(x) = f(g(x))$. $\bigcirc_{i=0}^{N} f_i$ means $f_N \circ f_{N-1} \circ \ldots \circ f_0$.
- Let $x \parallel y$ denote the concatenation of bit-strings x and y. If f and g are two functions, $f(x) \parallel g(y)$ is denoted by $f \parallel g$. $\Big\|_{i=0}^{N} f_i$ means $f_0 \parallel f_1 \parallel \ldots \parallel f_N$.

Appendix B.2 Details of the Implementation

The structures of the known white-box AES [1, 5] are static and the ShiftRows operation of each round is not included in the network of lookup tables. So, the boundaries of each round are obvious. To implement our method, first, we rearrange AES as follows:

$$AES = SR \circ ARK_{\hat{k}^{(10)}} \circ SB \circ ARK_{k^{(9)}} \circ \left(\bigcirc_{r=1}^{9} \left(MixC \circ SR \circ SB \circ ARK_{k^{(r-1)}} \right) \right), \tag{B1}$$

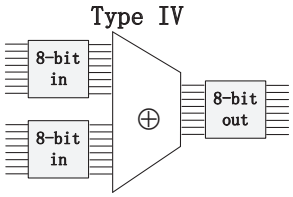

Figure B1 The 16-bit to 8-bit XOR table of Type IV.

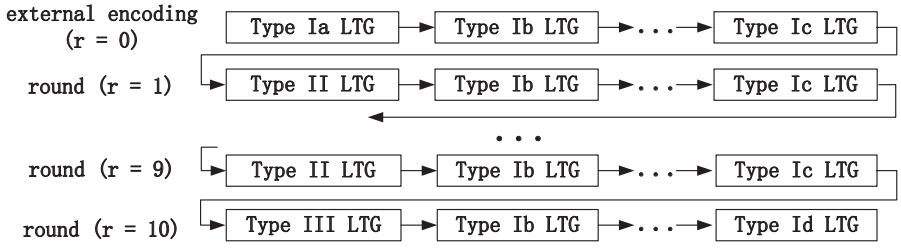

Figure B2 The whole structure of white-box AES with obfuscated round boundaries.

$\hat{k}^{(10)}$ is the result of applying ShiftRows to the round key $k^{(10)}$. Thus, we can combine SR and $MixC$ into one 128-bit to 128-bit linear transformation. Using the method mentioned in the main text, we can transform AES into a chain of LTGs. Here, $n = 128$ and $m = 8$. The mixing bijection MB and external encodings, F and G, are random 128×128 invertible matrices over $GF(2)$.

Each LTG includes 16 8-bit to 128-bit lookup tables followed by a layer of 15 XOR tables. Different with Chow et al.'s scheme, an XOR table of Type IV here is 16-bit to 8-bit, as shown in Figure B1. All tables are network-encoded by sets of random nonlinear 8-bit to 8-bit bijections.

The 8-bit to 128-bit lookup tables are generated in 6 different ways and named as Type Ia, Ib, Ic, Id, II and III. Then the whole structure of the WBC implementation can be shown as a chain of LTGs in Figure B2. In an LTG, all the 16 8-bit to 128-bit lookup tables are of the same type. The LTG of Type II or III is a K-LTG.

Suppose $(y_0, y_1, \ldots, y_{15})$ is an 16-byte input of an LTG, noted as $\Vert_{i=0}^{15} y_i$. The input and output encodings are $P_i^{(r)}$ and $Q_i^{(r)}$. Instead of combining two 4-bit to 4-bit nonlinear bijections with a 8-bit to 8-bit mixing bijection in Chow et al.'s scheme, $P_i^{(r)}$ and $Q_i^{(r)}$ here are just random 8-bit to 8-bit nonlinear bijections. We illustrate each LTG type as follows.

A **Type Ia LTG** is at the beginning of the chain and used to implement the external input encoding F. The main transformation is $MB_0^{(0)} \circ F$, $MB_0^{(0)}$ is a random protective mixing bijection. F is a 128×128 invertible matrix over $GF(2)$. The operation of multiplying with F can be implemented as XORing the results of 16 8-bit to 128-bit lookup tables F_i. Then the main transformation can be described as:

$$\left(MB_0^{(0)} \circ F\right)\left(\Vert_{i=0}^{15} y_i\right) = MB_0^{(0)}\left(\oplus_{i=0}^{15} F_i\left(y_i\right)\right) = \oplus_{i=0}^{15}\left(MB_0^{(0)}\left(F_i\left(y_i\right)\right)\right).$$

Considering the input and output encodings, each lookup table of Type Ia can be expressed as:

$$\left(\Vert_{j=0}^{15} Q_j\right) \circ MB_0^{(0)} \circ F_i \circ P_i,\ i = 0, 1, \ldots, 15,$$

i.e., includes an input encoding of nonlinear 8-bit to 8-bit bijection, an 8-bit to 128-bit transformation to compute F, a 128-bit to 128-bit mixing bijection $MB_0^{(0)}$ and 16 output encodings of nonlinear 8-bit to 8-bit bijection, as shown in Figure B3(a).

A **Type Ib LTG** is used to cancel the effect of the previous mixing bijection $MB_{previous}^{(r)}$ and add a new mixing bijection $MB_{current}^{(r)}$, as shown in Figure B3(b). A Type Ib lookup table can be expressed as:

$$\left(\Vert_{j=0}^{15} Q_j\right) \circ MB_{current}^{(r)} \circ \left(\left(MB_{previous}^{(r)}\right)^{-1}\right)_i \circ P_i,\ i = 0, 1, \ldots, 15.$$

Here, $\left(\left(MB_{previous}^{(r)}\right)^{-1}\right)_i$ is an 8-bit to 128-bit transformation and $\left(MB_{previous}^{(r)}\right)^{-1} = \oplus_{i=0}^{15}\left(\left(MB_{previous}^{(r)}\right)^{-1}\right)_i$. The sum of Type Ib LTGs in each round is random.

A **Type Ic LTG** is in front of a K-LTG, as shown in Figure B3(c). A Type Ic lookup table can be expressed as:

$$\left(\Vert_{j=0}^{15} Q_j\right) \circ \left(\left(MB_{previous}^{(r)}\right)^{-1}\right)_i \circ P_i,\ i = 0, 1, \ldots, 15.$$

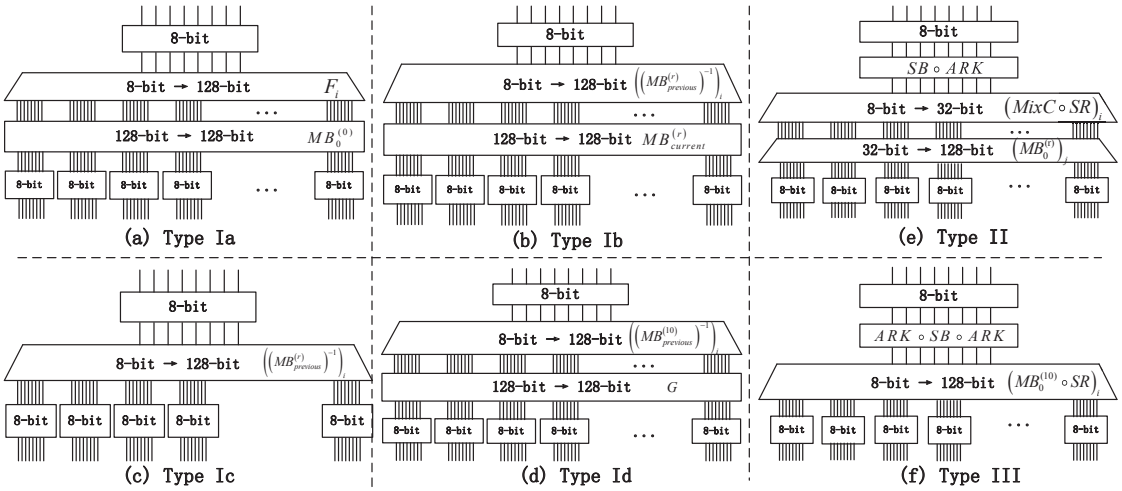

Figure B3 Six types of 8-bit to 128-bit lookup tables.

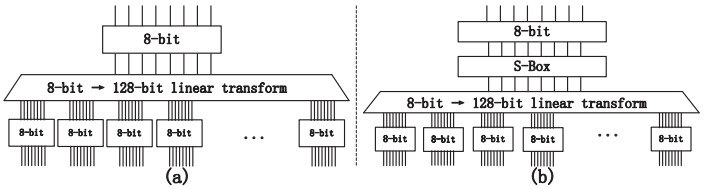

Figure B4 Two equivalent structures of the 8-bit to 128-bit lookup tables.

A **Type Id LTG** is at the end of the chain of LTGs in Figure B2 and used to implement the external output encoding G, as shown in Figure B3(d). A Type Id lookup table can be expressed as:

$$\left(\mathop{\Vert}_{j=0}^{15} Q_j\right) \circ G \circ \left(\left(MB_{previous}^{(10)}\right)^{-1}\right)_i \circ P_i,\ i = 0, 1, \ldots, 15.$$

A **Type II LTG** is at the beginning of round 1 to 9 in Figure B2, as shown in Figure B3(e). A Type II lookup table can be expressed as:

$$\left(\mathop{\Vert}_{k=0}^{15} Q_k\right) \circ \left(MB_0^{(r)}\right)_j \circ (MixC \circ SR)_i \circ (SB \circ ARK) \circ P_{(i,j)},\ i = 0, 1, 2, 3,\ j = 0, 1, 2, 3.$$

Here, $SB \circ ARK$ is an 8-bit to 8-bit transformation to compute the combination of AddRoundKey and SubBytes. $(MixC \circ SR)_i$ is an 8-bit to 32-bit transformation to compute the combination of ShiftRows and MixColumns and $MixC \circ SR = \mathop{\oplus}_{i=0}^{3} (MixC \circ SR)_i$. Similarly, $\left(MB_0^{(r)}\right)_j$ is a 32-bit to 128-bit transformation and $MB_0^{(r)} = \mathop{\oplus}_{j=0}^{3} \left(MB_0^{(r)}\right)_j$.

A **Type III LTG** is at the beginning of round 10, as shown in Figure B3(f). A Type III lookup table can be expressed as:

$$\left(\mathop{\Vert}_{j=0}^{15} Q_j\right) \circ \left(MB_0^{(10)} \circ SR\right)_i \circ (ARK \circ SB \circ ARK) \circ P_i,\ i = 0, 1, \ldots, 15.$$

Appendix B.3 Security Analysis

Appendix B.3.1 *Security in a black-box environment*

Leaving out the external encodings, the final output data produced by white-box AES implementation remain the same with standard AES. Thus, in a black-box environment, the security of white-box AES is at least as strong as standard AES.

Appendix B.3.2 *Analyzing the Structures of Lookup Tables*

The structure of Type Ia, Ib, Ic and Id lookup tables can be equivalently simplified as Figure B4(a), which includes a random nonlinear 8-bit to 8-bit bijections, an 8-bit to 128-bit transformation to compute the mixing bijection and 16 output encodings of nonlinear 8-bit to 8-bit bijection. Comparing to Figure B4(a), in the simplified structure of Type II and III, there is an extra key-dependent 8-bit to 8-bit S-box, as shown in Figure B4(b). Because all of the bijections are unknown to the attacker, the two structures are actually the same and hard to be distinguished.

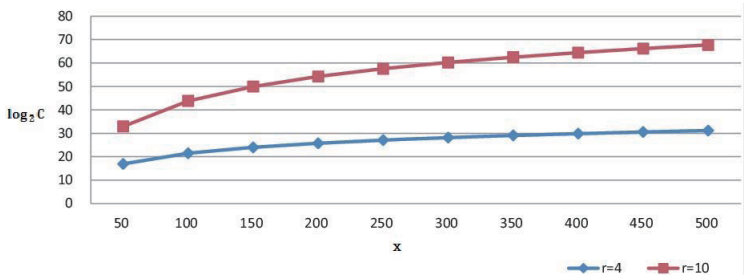

Figure B5 The work factor $C = \mathcal{C}^r_{x-(10-r)-1}$ increases when x increases, $r = 4, 10$.

Table B1 White-box AES implementations under know attacks.

Implementation	Known attacks	Fastest attack efficiency
[1]	[2] [3]	2^{22}
[4]	[6]	2^{17}
[5]	[7]	2^{32}
RBO-WBAES(10, 500)	[2] [3]	2^{53}
RBO-WBAES(10, 500) (with irreversible key scheduling)	[2] [3]	2^{90}

Table B2 Performances of white-box AES implementations.

Implementation	Total size	Times of doing table lookups
[13]	1.25KB	304 (need extra 119 XORs)
[1]	752KB	3104
[5]	20502KB	80 (need extra 40 XORs, 11 128×128 matrix multiplications)
RBO-WBAES(10, 500)	510MB	15810

Appendix B.3.3 *Analyzing the Structures of LTGs*

Considering the lookup tables in an LTG as a whole, both K-LTGs and other LTGs can collapse to the structure of SAS. It is hard for an attacker to distinguish them from each other by just figuring out their structures. But an LTG of structure SAS could be decomposed [14]. The attacker can obtain an equivalent representation of all the secret layers and treat the whole compressed version as a large secret key. However, as mentioned in section Appendix A.2, because of the randomly selected bijections (MB,P_i and Q_j) in the LTGs, our implementation follows **protocol-assisted incompressibility**.

Appendix B.3.4 *Implementing Known Attacks*

The key scheduling algorithm of AES is invertible. One known round key is adequate to recover the main key. Among the known attacks towards Chow et al.'s white-box AES, Lepoint et al.'s attack [3] with a work factor of 2^{22} is the most efficient. In their attack, 3 consecutive rounds are attacked to get one round key. Although the input and output encodings are changed to 8-bit to 8-bit bijections in our implementation, Lepoint et al.'s attack still works. To apply Lepoint et al.'s attack, 4 consecutive K-LTGs should be located.

In the chain of LTGs, there are 10 K-LTGs. According to the main text, the work factor to determine the first 4 K-LTGs is $C = \mathcal{C}^4_{x-(10-4)-1} = \mathcal{C}^4_{x-7}$, x is the total sum of LTGs other than K-LTGs. Along with the increasing of x, C increases, as shown in Figure B5.

From Figure B5, we can see that when $x = 500$, the work factor of confirming the round boundaries is $\mathcal{C}^4_{500-7} \approx 2^{31}$. Then, to implement Lepoint et al.'s attack, the total attack work factor will be increased to a practical security level of $2^{31} \times 2^{22} = 2^{53}$. Furthermore, if the key scheduling algorithm of AES is improved to an irreversible one, e.g., in [11, 12], the total attack work factor will be increased to $\mathcal{C}^{10}_{500-1} \times 2^{22} \approx 2^{90}$, as shown in Figure B5 with the case of $r = 10$. In the actual application, if a higher level of security is needed, a bigger x could be chosen.

Table B1 shows the comparison of white-box AES implementations under known attacks. Our white-box AES with obfuscated round boundaries is abbreviated to "RBO-WBAES(10, 500)".

Appendix B.4 Size and Performance

Table B2 shows the comparison of the performances of lookup-table-based white-box AES implementations. The running speeds can be estimated by the times of doing table lookups.

From table B2, we can see that RBO-WBAES(10, 500) sacrifice efficiency to get high security level. In order to get a more intuitive impression, we tested the performance of RBO-WBAES(10, 500) in an environment: Intel Xeon CPU

E5-2620 2G, 16G RAM and 64-bit Windows Server 2008 R2. It takes 1.752ms to complete encrypting a block of 16-byte data. On average, it takes 0.0034ms to complete the operation of one LTG.

WBC implementations are suitable in such environments: **(i)** white-box attack model; **(ii)** the frequency of rekeying is not high; **(iii)** the data which is to be conducted is not big, e.g., an encrypted 128-bit session key. If a WBC implementation is used to conduct big data, parallel programming is necessary. Actually, take our RBO-WBAES(10, 500) for example, a pipeline optimization algorithm can be designed. After the using of one LTG, it can be used to conduct the next block immediately. Thus, the speed of conduct big data can be estimated by the time consuming of completing one LTG's operation. In our test, to encrypt 1MB data, it will take approximately 0.225s.

References

1 Chow S, Eisen P, Johnson H, et al. White-box cryptography and an AES implementation. In: International Workshop on Selected Areas in Cryptography. Berlin: Springer, 2003. 250–270

2 Billet O, Gilbert H, Ech-Chatbi C. Cryptanalysis of a white box AES implementation. In: International Workshop on Selected Areas in Cryptography. Berlin: Springer, 2005. 227–240

3 Lepoint T, Rivain M, De Mulder Y, et al. Two attacks on a white-box AES implementation. In: International Conference on Selected Areas in Cryptography. Berlin: Springer, 2014. 265–285

4 Bringer J, Chabanne H, Dottax E. White box cryptography: another attempt. IACR Cryptology ePrint Archive, 2006, 2011: 468

5 Xiao Y Y, Lai X J. A secure implementation of white-box AES. In: Proceedings of the 2nd International Conference on Computer Science and its Applications, Jeju, 2009. 1–6

6 De Mulder Y, Wyseur B, Preneel B. Cryptanalysis of a perturbated white-box AES implementation. In: Progress in Cryptology-INDOCRYPT. Berlin: Springer, 2010. 292–310

7 De Mulder Y, Roelse P, Preneel B. Cryptanalysis of the XiaoCLai white-box AES implementation. In: International Conference on Selected Areas in Cryptography. Berlin: Springer, 2013. 34–49

8 Biryukov A, Bouillaguet C, Khovratovich D. Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key. In: International Conference on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2014. 63–84

9 Dinur I, Dunkelman O, Kranz T, et al. Decomposing the ASASA Block Cipher Construction. IACR Cryptology ePrint Archive, 2015: 507

10 Bogdanov A, Isobe T. White-box cryptography revisited: space-hard ciphers. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. New York: ACM, 2015. 1058–1069

11 Bai K, Wu C. An AES-Like Cipher and Its White-Box Implementation. The Computer Journal, 2016, 59(7): 1054–1065

12 Xu T, Liu F, Wu C. A white-box AES-like implementation based on key-dependent substitution-linear transformations. Multimedia Tools and Applications, 2017: 1–21

13 Pub N F. 197: Advanced encryption standard (AES). Federal Information Processing Standards Publication, 2001, 197(441): 0311

14 Biryukov A, Shamir A. Structural cryptanalysis of SASAS. In: International Conference on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2001. 395–405