Cryptanalysis of a Type of White-Box Implementations of the SM4 Block Cipher

Jiqiang Lu, Jingyu Li

Abstract—The SM4 block cipher was first released in 2006 as SMS4 used in the Chinese national standard WAPI, and became a Chinese national standard in 2016 and an ISO international standard in 2021. White-box cryptography aims primarily to protect the secret key used in a cryptographic software implementation in the white-box scenario that assumes an attacker to have full access to the execution environment and execution details of an implementation. Since white-box cryptography has many real-life applications nowadays, a few white-box implementations of the SM4 block cipher has been proposed with its increasingly wide use, among which a type of constructions is dominated, that use an affine diagonal block encoding to protect the original XOR sum of the three branches entering the S-box layer of a round and use its inverse to protect the original input of the S-box layer, such as Xiao and Lai’s implementation in 2009, Shang’s implementation in 2016 and Yao and Chen’s implementation in 2020. In this paper, we show that this type of white-box SM4 constructions can be somewhat equivalent to a plain implementation mostly with Boolean masks from a security viewpoint, by devising collision-based attacks on Xiao and Lai’s, Shang’s and Yao and Chen’s implementations with a time complexity of respectively about $2^{39}$, $2^{32}$ and $2^{32}$ to peel off most white-box operations until only Boolean masks remain. Besides, we present a collision-based attack on a white-box SM4 implementation with a time complexity of about $2^{37}$ to recover an original round key, which uses a linear diagonal block encoding instead of an affine diagonal block encoding. Our results show that generating such a white-box SM4 implementation with affine encodings can be simplified into generating a plain implementation with Boolean masks (if its security expectation is beyond the above-mentioned complexity), and the effect of an affine encoding is significantly better than the effect of a linear encoding in the sense of our cryptanalysis results.

Index Terms—White-box cryptography, SM4 (SMS4) block cipher, collision attack.

I. INTRODUCTION

In 2002, Chow et al. [10], [11] introduced white-box cryptography and proposed white-box implementations of the AES [29] and DES [30] block ciphers. White-box cryptography works under the white-box security model, which assumes an attacker has full access to the execution environment and execution details (such as intermediate values, CPU calls, memory registers, etc) of a software implementation, giving the attacker more power than the black-box and grey-box security models. Nowadays, white-box cryptography has many real-life application scenarios like TV boxes, mobile phones and game consoles, and some white-box cryptography solutions have been in use.

The primary security threat for white-box cryptography is key extraction attack, which aims to extract the key used in white-box implementation. Chow et al.’s white-box AES implementation has been cryptanalysed extensively [6], [22], [28], [33], and the main attack results are as follows. In 2004, Billet et al. [6] presented an attack with a time complexity of $2^{30}$ (referred to below as BGE attack). In 2013, Lepoint et al. [22] improved the BGE attack to have a time complexity of $2^{22}$, and presented a collision-based attack with a time complexity of $2^{22}$. There are also a few attacks [18], [19], [25], [36] on Chow et al.’s white-box DES implementation. On the other hand, a number of different white-box implementation designs have been proposed [1], [3], [9], [20], [27], [37], but almost all of them have been broken with a practical or semi-practical time complexity [3], [12]–[14], [22]. Generally speaking, it has been well understood that the line of white-box implementation for an existing cryptographic algorithm is hardly possible to achieve the full security under the black-box model, but it is expected that it can still provide some protection with realistic significance.

The SM4 block cipher was first released in 2006 as the SMS4 [15] block cipher used in the Chinese national standard WAPI (WLAN Authentication and Privacy Infrastructure), which has a 128-bit block length and a 128-bit user key with a total of 32 rounds. SMS4 became a Chinese cryptographic industry standard in 2012, labeled with SM4, which then became a Chinese national standard [16] in 2016 and an ISO international standard in 2021 [17]. The main white-box implementation results of SMS4/SM4 are as follows. In 2009, Xiao and Lai [38] proposed the first white-box SM4 implementation in a relatively traditional way with a series of lookup tables and affine transformations. In 2013, Lin and Lai [23] attacked Xiao and Lai’s white-box SM4 implementation with a time complexity of around $2^{27}$ by combining the BGE attack with other techniques like differential cryptanalysis [5]. In 2015, Shi et al. [32] proposed a lightweight white-box SM4 implementation based on the idea of dual cipher [4]. In 2016, Shang [31] improved Xiao and Lai’s white-box SM4 implementation mainly by merging two individual lookup tables for two S-boxes into a larger whole, and got a security complexity of around $2^{48}$; and Bai and Wu [2] proposed a white-box SM4 implementation with an S-
box input being divided into two shares. In 2018, Lin et al. [24] applied Biryukov et al.’s affine equivalence technique [7] to attack Shi et al.’s white-box SM4 implementation with a time complexity of $2^{49}$. In 2020, Yao and Chen [39] proposed a white-box SM4 implementation with some original internal states expanded by dummy states under the control of a secret random number, and got the lowest attack complexity of about $2^{51}$ among a variety of attack techniques; and Wu et al. [35] proposed a white-box SM4 implementation with lookup tables and linear transformations, and showed it was resistant against BGE attack. In 2021, Wang et al. [34] applied Lepoint et al.’s collision-based idea to attack Shi et al.’s white-box SM4 implementation with a time complexity of about $2^{22}$ (note that this time complexity is obtained for checking the first-order derivatives for all $2^7$ inputs under the worst case, and the time complexity under the expected case would be around $2^{17.1}$, as we do for Wu et al.’s white-box SM4 implementation in Section VI).

In this paper, we are concerned with Xiao and Lai’s, Shang’s, Yao and Chen’s and Wu et al.’s white-box SM4 implementations, which are more or less different one another from a structural view but fundamentally all employ the construction method that uses an affine (or extremely even linear) diagonal block encoding to protect the original output of the XOR sum of the three branches entering the S-box layer of a round and uses the inverse of the encoding to protect the original input of the S-box layer. Especially, we focus on Yao and Chen’s white-box SM4 implementation due to its representativeness, and apply Lepoint et al.’s collision-based idea to devise an attack with a total time complexity of about $2^{22}$ to peel off most white-box operations until only Boolean masks remain (but it seems impossible to work out the original key from the masked round keys), where the Boolean mask for a round key is associated with the affine diagonal block encoding; in particular, we first find that the effect of those dummy states can be bypassed without any workload by devising an appropriate collision function, then we find a trick to recover the linear parts of the affine diagonal block encodings at ease, next we recover the masked round key, and finally we recover both the linear (and constant sometimes) parts of the general affine encodings used. With more or less modifications due to their respective specifications, the attack is similarly applied to Xiao and Lai’s and Shang’s white-box SM4 implementations with a time complexity of about $2^{22}$ and $2^{39}$, respectively, but can recover the original round key for Wu et al.’s white-box SM4 implementation with a time complexity of about $2^{17.1}$, since Wu et al.’s implementation uses a linear diagonal block encoding instead of an affine diagonal block encoding and so there is no mask at last. Our results show that generating such a white-box SM4 construction with affine encodings can be simplified by generating a plain implementation mostly with Boolean masks to avoid the cumbersome affine encodings, if its security expectation is beyond the above-mentioned complexity; and the effect of an affine encoding is significantly better than the effect of a linear encoding.

The remainder of the paper is organised as follows. We describe the notation and the SM4 block cipher in the next section, and present our cryptanalysis results on Yao and Chen’s, Xiao and Lai’s, Shang’s and Wu et al.’s white-box SM4 implementations in Sections III to VI, respectively. Section VII concludes this paper.

II. PRELIMINARIES

In this section, we give the notation used throughout this paper, and briefly describe the SM4 block cipher.

A. Notation

We use the following notation throughout this paper.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊕</td>
<td>bitwise exclusive OR (XOR)</td>
</tr>
<tr>
<td>↗</td>
<td>right shift of a bit string</td>
</tr>
<tr>
<td>⋅</td>
<td>left rotation of a bit string</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>◦</td>
<td>functional composition</td>
</tr>
</tbody>
</table>

B. The SM4 Block Cipher

SM4 [15], [16] is a generalised Feistel cipher with 32 rounds, a 128-bit block size and a 128-bit key length. Denote by $(X_i, X_{i+1}, X_{i+2}, X_{i+3})$ the 128-bit input to the $i$-th round, by $rk_i$ the 32-bit $i$-th round key, where $X_i \in \mathbb{GF}(2)^{32}$ and $i = 0, 1, \ldots, 31$.

Define the nonlinear function $\tau : \mathbb{GF}(2)^{32} \rightarrow \mathbb{GF}(2)^{32}$ that applies the same 8-bit S-box $S$ four times in parallel as

$$ x \mapsto (S(x_{[31\ldots24]}), S(x_{[23\ldots16]}), S(x_{[15\ldots8]}), S(x_{[7\ldots0]})) ; $$

and define the linear function $L : \mathbb{GF}(2)^{32} \rightarrow \mathbb{GF}(2)^{32}$ as

$$ x \mapsto x \oplus (x \ll 2) \oplus (x \ll 10) \oplus (x \ll 18) \oplus (x \ll 24). \tag{1} $$

Then, the invertible transformation $T : \mathbb{GF}(2)^{32} \times \mathbb{GF}(2)^{32} \rightarrow \mathbb{GF}(2)^{32}$ is defined to be

$$ (x, rk_i) \rightarrow L(\tau(x \oplus rk_i)), $$

and the round function $F : \mathbb{GF}(2)^{128} \times \mathbb{GF}(2)^{32} \rightarrow \mathbb{GF}(2)^{128}$ under round key $rk_i$ is

$$ ((X_i, X_{i+1}, X_{i+2}, X_{i+3}), rk_i) \mapsto (X_{i+1}, X_{i+2}, X_{i+3}, X_i \oplus T(X_{i+1} \oplus X_{i+2} \oplus X_{i+3} \oplus rk_i)). \tag{2} $$

The encryption procedure of SM4, as depicted in Fig. 1, consists of the 32 round functions $F$’s and finally a reverse transformation $R : \mathbb{GF}(2)^{128} \rightarrow \mathbb{GF}(2)^{128}$ defined as

$$ (X_{32}, X_{33}, X_{34}, X_{35}) \mapsto (X_{35}, X_{34}, X_{33}, X_{32}). $$

The decryption process of SM4 is the same as the encryption process, except that the round keys are used in the reverse order. We refer the reader to [15], [16] for detailed specifications.

Particularly, it is easy and worthy to note that the linear transformation $L$ (as described in Eq. (1)) of SM4 can also be represented as an invertible $32 \times 32$-bit matrix

$$
\begin{bmatrix}
B_1 & B_2 & B_2 & B_3 \\
B_3 & B_1 & B_2 & B_2 \\
B_2 & B_3 & B_1 & B_2 \\
B_2 & B_2 & B_3 & B_1 \\
\end{bmatrix},
$$

(3)
with $B_1, B_2$ and $B_3$ being invertible $8 \times 8$-bit block matrices.

Let $x_0, x_1, x_2, x_3$ be four byte variables, represent $L$ as four $32 \times 8$-bit matrices $[L_0 \ L_1 \ L_2 \ L_3]$, and define
\[
L_0(x) = x \cdot [B_1 \ B_3 \ B_2 \ B_1]^T,
\]
\[
L_1(x) = x \cdot [B_3 \ B_1 \ B_3 \ B_2]^T,
\]
\[
L_2(x) = x \cdot [B_2 \ B_2 \ B_1 \ B_3]^T,
\]
\[
L_3(x) = x \cdot [B_3 \ B_2 \ B_2 \ B_1]^T,
\]
then $L(x_0|x_1|x_2|x_3) = L_0(x_0) \oplus L_1(x_1) \oplus L_2(x_2) \oplus L_3(x_3)$.

III. Cryptanalysis of Yao and Chen’s White-Box SM4 Implementation

In this section, we first describe Yao and Chen’s white-box SM4 implementation, and then present our cryptanalysis result on it.

A. Yao and Chen’s White-Box SM4 Implementation

Yao and Chen’s white-box SM4 implementation [39] is based on internal state expansion, particularly, the $32 \times 32$-bit matrix representation described in Eq. (3) of the linear transformation $L$ is expanded to the following $64 \times 64$-bit matrix $\tilde{L}$ with the $8 \times 8$-bit zero matrix $0$:
\[
\tilde{L} = \begin{bmatrix} B_1 & 0 & B_2 & 0 & B_2 & 0 & B_3 & 0 \\ 0 & B_1 & 0 & B_2 & 0 & B_2 & 0 & B_3 \\ B_1 & 0 & B_2 & 0 & B_2 & 0 & B_3 & 0 \\ 0 & B_2 & 0 & B_3 & 0 & B_3 & 0 & B_1 \\ B_2 & 0 & B_3 & 0 & B_1 & 0 & B_2 & 0 \\ 0 & B_2 & 0 & B_3 & 0 & B_1 & 0 & B_2 \\ B_3 & 0 & B_1 & 0 & B_2 & 0 & B_2 & 0 \\ 0 & B_3 & 0 & B_1 & 0 & B_2 & 0 & B_2 \end{bmatrix}.
\]

Represent the matrix $\tilde{L}$ as four $64 \times 16$-bit matrices, that is, $\tilde{L} = [\tilde{L}_0 \ \tilde{L}_1 \ \tilde{L}_2 \ \tilde{L}_3]$. Then, an encryption round of Yao and Chen’s white-box SM4 implementation consists of the following three parts according to Eq. (2), as depicted in Fig. 2.

Note first that $X_i$ is the corresponding original value protected with an affine output encoding $P_i(x) = A_i \cdot x + a_i$, where $x$ is a 32-bit variable, the linear part $A_i$ is a secret (randomly generated) general invertible $32 \times 32$-bit matrix, the constant part $a_i$ is a secret (randomly generated) 32-bit vector, and $i = 0, 1, \ldots, 35$.

1) Part 1 – Implement $X_{i+1}^l + X_{i+2} + X_{i+3} \rightarrow Z_i$. In order to obtain the original value of $X_{i+1}^l + X_{i+2} + X_{i+3}$ from the protected forms $X_{i+1}^l, X_{i+2}$ and $X_{i+3}$, apply first the inverses $P_{i+1}^{-1}, P_{i+2}^{-1}$ and $P_{i+3}^{-1}$ of the three output encodings respectively to $X_{i+1}^l, X_{i+2}$ and $X_{i+3}$, followed by an identical diagonal output encoding $E_i = \text{diag}(E_{i,0}, E_{i,1}, E_{i,2}, E_{i,3})$, where $E_{i,0}, E_{i,1}, E_{i,2}, E_{i,3}$ are four general invertible $8 \times 8$-bit affine transformations ($i = 0, 1, \ldots, 31$).

This part can be summarised as
\[
X_{i+1}^l = E_i \circ P_{i+1}^{-1}(X_{i+1}^l), \quad j = 1, 2, 3;
\]
\[
Z_i = X_{i+1}^l + X_{i+2} + X_{i+3},
\]
where $Z_i$ is a 32-bit variable. Observe that the final result of this part $Z_i = E_i \circ P_{i+1}^{-1}(X_{i+1}^l) + P_{i+2}^{-1}(X_{i+2}) + P_{i+3}^{-1}(X_{i+3})$ is the original value of $X_{i+1} + X_{i+2} + X_{i+3}$ protected with the output encoding $E_i$ in such a way that its four bytes are protected respectively with the four $8$-bit encodings $E_{i,0}, E_{i,1}, E_{i,2}, E_{i,3}$.

2) Part 2 – Implement $T(Z_i, r_k^i) \rightarrow Y_i = Y_i^l || Y_i^r$: The input $Z_i$ of the second part is the output of the first part, represent $Z_i$ as $4$ bytes $Z_i = (z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$, and represent the round key $r_k^i$ as $4$ bytes $r_k^i = (r_{k,0}^i, r_{k,1}^i, r_{k,2}^i, r_{k,3}^i)$, where $i = 0, 1, \ldots, 31$. Next, construct four lookup tables that map from 8-bit input to 64-bit output each, as follow:

\[
\begin{align*}
Table_{e,0} & = G_{i,0} \circ \tilde{L}_0 \circ \tilde{S}(E_{i,0}^{-1}(z_{i,0}), r_{k,0}^i, \alpha_{i,j}^0, 0), \\
Table_{e,1} & = G_{i,1} \circ \tilde{L}_1 \circ \tilde{S}(E_{i,1}^{-1}(z_{i,1}), r_{k,1}^i, \alpha_{i,1}^1, 0), \\
Table_{e,2} & = G_{i,2} \circ \tilde{L}_2 \circ \tilde{S}(E_{i,2}^{-1}(z_{i,2}), r_{k,2}^i, \alpha_{i,2}^2, 0), \\
Table_{e,3} & = G_{i,3} \circ \tilde{L}_3 \circ \tilde{S}(E_{i,3}^{-1}(z_{i,3}), r_{k,3}^i, \alpha_{i,3}^3, 0),
\end{align*}
\]
where
\[
\begin{align*}
\tilde{S}(E_{i,j}^{-1}(z_{i,j}), r_{k,j}, \alpha_{i,j}) & = \begin{cases} \\
\tilde{S}(E_{i,j}^{-1}(z_{i,j}) + r_{k,j}, \alpha_{i,j}) = \tilde{S}(E_{i,j}^{-1}(z_{i,j}) + r_{k,j}) & t_{i,j} = 0, \\
\tilde{S}(E_{i,j}^{-1}(z_{i,j}) + r_{k,j}, \alpha_{i,j}) = \tilde{S}(E_{i,j}^{-1}(z_{i,j}) + r_{k,j}) & t_{i,j} = 1.
\end{cases}
\end{align*}
\]

That is, the $\tilde{S}$ operation is constructed by expanding the original $S$ operation with a dummy $S$ operation under the control of the 1-bit $t_{i,j}$ parameter.
• $G_{i,j}$ is the composition of a shift matrix $SR$ and an output encoding $Q_{i,j}$. The shift matrix $SR$ transforms the expanded 64-bit value after $L_j$ into such a 64-bit value that the former half is the original 32-bit part (without expansion) and the latter half consists only of some dummy bits. $Q_{i,j}$ is of the affine form $Q_{i,j}(x) = L_{Q_i} \cdot x \oplus C_{Q_{i,j}}$, here $x$ is a 64-bit variable, the linear part $L_{Q_i}$ is a block diagonal matrix being composed of eight $8 \times 8$-bit matrices, and the constant part $C_{Q_{i,j}}$ consists of eight concatenated 8-bit vectors.

The final output of this part is the XOR of the four 64-bit outputs of the four lookup tables, which is denoted by $Y_i = Y_{i0} || Y_{i1}$ with $Y_{i0}$ being supposed to be the original useful 32-bit value.

3) Part 3 – Implement $Y_{i0} \oplus X_i \mapsto X_{i+4}$: This part first extracts the original useful 32-bit value from the 64-bit expanded output of the second part, and then calculates $X_{i+4}$, as follows.

$$
Y'_{i0} = P''_{i+4} \circ (Q_i)^{-1}(Y_{i0}),
$$

$$
X'_i = P''_{i+4} \circ P^{-1}_i(X_i),
$$

$$
X_{i+4} = Y'_{i0} \oplus X'_i,
$$

where $(Q_i)^{-1}$ represents the corresponding part of the inverse of the encodings $L_{Q_i} : x \oplus (C_{Q_{i+1}} \oplus C_{Q_{i+2}} \oplus C_{Q_{i+3}}) \oplus C_{Q_{i,0}}$ of the second part, and $P''_{i+4}$ and $P''_{i-4}$ are new affine output encodings of the forms $P''_{i+4}(x) = P_{i+4} \circ a_{i+4}'$ and $P''_{i-4}(x) = P_{i-4} \circ a_{i-4}'$, respectively, so that $X_{i+4}$ is a protected form with an affine output encoding $P''_{i+4}(x) = A_{i+4} \cdot x \oplus a_{i+4}$, like $X_i$.

As a result, the whole white-box SM4 implementation can be obtained by iterating the above process for all the 32 rounds with possibly independent encodings.

Yao and Chen analysed its security against a variety of attack techniques like BGE, and got that the attack complexity was $2^{97}$, and the lowest attack complexity was $2^{51}$ among all used attack techniques.

B. Cryptanalysis of Yao and Chen’s Implementation

In this subsection, we apply Lepoint et al.’s collision-based attack idea to peel off most white-box operations until only Boolean masks remain for Yao and Chen’s white-box SM4 implementation, with a time complexity of about $2^{24.4}$. AES and SM4 have different structures, and Yao and Chen’s white-box SM4 implementation is distinct from Chow et al.’s white-box AES implementation: there are dummy states with indeterminate positions and the encoding used in $X_{i+4}$ involves a general $32 \times 32$-bit matrix, which does not allow us to apply Lepoint et al.’s attack idea efficiently within one round, as for Chow et al.’s white-box AES implementation. However, after a detailed investigation we find an appropriate collision function by considering two consecutive rounds in Yao and Chen’s white-box SM4 implementation, plus a trick that can recover the linear parts of the concerned encodings, to bypass the effects due to the dummy states and etc.

1) Devising a Collision Function: As illustrated in Fig. 3 at a high level, the collision function used in our attack takes as input the two 32-bit input parameters $(z_{i,0} || z_{i,1} || z_{i,2} || z_{i,3}, X_i)$ in the second part of an encryption round of Yao and Chen’s white-box SM4 implementation, and ends with the output of an $E_{i+1,j}$ operation of the $X_{i+4}$ branch in the first part of the next encryption round ($j = 0, 1, 2, 3$). Observe that $E_i$ and $E_{i+1}$ are diagonal affine transformations, $E_{i,j}$ and $E_{i+1,j}$ are invertible $8 \times 8$-bit affine transformations, and $z_{i,j}$ is the original input byte to the $j$-th original S-box of the $i$-th encryption round in a protected form with $E_{i,j}$.

The collision function is functionally equivalent and can be simplified to the one depicted in Fig. 4. In our attack and all subsequent descriptions, we set $X_i$ such that $P'_{i+4} \circ P^{-1}_i(X_i) = 0$, and denote the constant $A_{i+4}'$, $a_{i+4}' \oplus A_{i+4}' \circ P'_{i+4} \circ P^{-1}_i(X_i) = A_{i+4}' \circ a_{i+4}'$ by $\varepsilon_i$. We now explain where the value $\varepsilon_i$ comes from. Let $\hat{Y}_i$ denotes the original 32-bit value immediately after the $L$ operation under the input $Z_i = (z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$, then we have

$$
P'_{i+4} \circ (Y'_{i0} \oplus P''_{i+4} \circ P^{-1}_i(X_i)) = P'_{i+4} \circ (Y'_{i0} \oplus A_{i+4} \circ P''_{i+4} \circ P^{-1}_i(X_i)) = P'_{i+4} \circ P''_{i+4}(\hat{Y}_i) = P^{-1}_{i+4} \circ (P_{i+4}(\hat{Y}_i) \oplus a_{i+4}') = P^{-1}_{i+4} \circ (A_{i+4}(\hat{Y}_i) \oplus a_{i+4} + a_{i+4}') = A^{-1}_{i+4} \circ (A_{i+4}(\hat{Y}_i) \oplus a_{i+4} + a_{i+4}') = \hat{Y}_i \oplus A^{-1}_{i+4} \circ a_{i+4}',
$$

which is equal to $\hat{Y}_i \oplus \varepsilon_i$ under $P'_{i+4} \circ P^{-1}_i(X_i) = 0$.

As a consequence, the collision function denoted by $f^i(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3}, X_i)$, or simply $f^i(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$ under $P'_{i+4} \circ P^{-1}_i(X_i) = 0$, is

$$
f^i(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3}) = \left[ \begin{array}{c} E_{i+1,0} \\ E_{i+1,1} \\ E_{i+1,2} \\ E_{i+1,3} \end{array} \right] \circ \oplus_{\varepsilon_i} \circ L \circ \left[ \begin{array}{c} S \circ \oplus_{k_{i,0}} \circ E_{i-1}^{-1}(z_{i,0}) \\ S \circ \oplus_{k_{i,1}} \circ E_{i-1}^{-1}(z_{i,1}) \\ S \circ \oplus_{k_{i,2}} \circ E_{i-1}^{-1}(z_{i,2}) \\ S \circ \oplus_{k_{i,3}} \circ E_{i-1}^{-1}(z_{i,3}) \end{array} \right].
$$

Furthermore, we express $f^i$ as a concatenation of four byte functions $f_0^i$, $f_1^i$, $f_2^i$ and $f_3^i$:

$$
f^i(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3}) = \left[ f_0^i(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3}), f_1^i(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3}), f_2^i(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3}), f_3^i(z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3}) \right]^T;
$$

Figure 3. Our collision function on Yao and Chen’s white-box SM4 implementation.
and define $S_j$ function as

$$S_j^i(\cdot) = S \circ rk_{i,j} \circ E^{-1}_{i,j}(\cdot) = S(rk_{i,j} \circ E^{-1}_{i,j}(\cdot)), \quad j = 0, 1, 2, 3. \quad (4)$$

2) Recovering $S_j^i$ Functions: Next we try to recover the functions $S_0$, $S_1$, $S_2$, and $S_3$ by exploiting collisions on the output of the functions $f_j^i$. We first use the following collision to recover $S_0^i$ and $S_0^i$:

$$f_0^i(\alpha, 0, 0, 0) = f_0^i(0, \beta, 0, 0), \quad (5)$$

where $\alpha, \beta \in \text{GF}(2)^8$. By the linear transformation $L$ in Eq. (3), Eq. (5) immediately means the following equation:

$$E_{i+1,0} \circ \epsilon_{i,0} \circ (B_1 \circ S_0^i(\alpha) + B_2 \circ S_1^i(0)) + B_2 \circ S_2^i(0) + B_3 \circ S_3^i(0)) = E_{i+1,0} \circ \epsilon_{i,0} \circ (B_1 \circ S_0^i(0) + B_2 \circ S_1^i(\beta) + B_2 \circ S_2^i(0) + B_3 \circ S_3^i(0)),$$

where $\epsilon_{i,0}$ is the corresponding byte of the constant $\epsilon_i$. Since $E_{i+1,0}$ is a bijection, we have the following equation:

$$B_1 \circ S_0^i(\alpha) + B_2 \circ S_1^i(0) = B_1 \circ S_0^i(0) + B_2 \circ S_1^i(\beta).$$

For convenience, define $u_m = S_0^i(m)$ and $v_m = S_1^i(m)$, then we have

$$B_1 \circ (u_0 + u_\alpha) = B_2 \circ (v_0 + v_\beta). \quad (6)$$

Since $\alpha \mapsto f_0^i(\alpha, 0, 0, 0)$ and $\beta \mapsto f_0^i(0, \beta, 0, 0)$ are bijections, we can find 256 collisions. After removing $\epsilon_{i,0} = (0, 0)$, we get 255 pairs $(\alpha, \beta)$ satisfying Eq. (5), each providing an equation of the form of Eq. (6). In the same way, we use other $f_j^i$ functions ($j \in \{1, 2, 3\}$) to generate similar equations with different coefficients in $\{B_1, B_2, B_3\}$. Finally, we get $4 \times 255$ linear equations with all 512 unknowns, as follows:

$$\begin{align*}
B_1 \circ (u_0 + u_\alpha) &= B_2 \circ (v_0 + v_\beta); \\
B_3 \circ (u_0 + u_\alpha) &= B_1 \circ (v_0 + v_\beta); \\
B_2 \circ (u_0 + u_\alpha) &= B_3 \circ (v_0 + v_\beta); \\
B_2 \circ (u_0 + u_\alpha) &= B_2 \circ (v_0 + v_\beta).
\end{align*} \quad (7)$$

Define $u_m' = u_0 + u_m$ and $v_m' = v_0 + v_m$, with $m \in \{1, 2, \ldots, 255\}$, so that the number of unknowns is reduced to $2 \times 255 = 510$. Thus, Eq. (6) can be rewritten as

$$B_1 \circ u_m' = B_2 \circ v_\beta,$$

meaning that the linear system of Eq. (7) can be represented with 510 unknowns as

$$\begin{align*}
B_1 \circ u_m' &= B_2 \circ v_j', \\
B_3 \circ (u_0 + u_\alpha) &= B_1 \circ v_j', \\
B_2 \circ (u_0 + u_\alpha) &= B_3 \circ v_j', \\
B_2 \circ (u_0 + u_\alpha) &= B_2 \circ v_j'.
\end{align*}$$

The $4 \times 255$ equations yield a linear system of rank 509; and in such a linear equation system, all other unknowns can be expressed as a function of one of them, say $u_1'$, that is, there exist coefficients $a_i$ and $b_i$ such that $u_m' = a_m \cdot u_1'$ and $v_m' = b_m \cdot u_1'$. That is,

$$\begin{align*}
u_M &= a_m \cdot (u_0 + u_1) + u_0, \\
v_M &= b_m \cdot (u_0 + u_1) + v_0.
\end{align*} \quad (8)$$

Next we can recover the $S_0^i$ function by exhaustive search on the pair $(u_0, u_1)$, and at last we use the following equation from the definition of the $S_0^i$ function to verify whether the obtained $S_0^i$ function is right or not:

$$S^{-1} \circ S_0^i(\cdot) = rk_{i,0} \circ E^{-1}_{i+1,0}(\cdot).$$

Since $E^{-1}_{i+1,0}$ is an $8 \times 8$-bit invertible affine transformation, the above function has an algebraic degree of at most 1. For a wrong pair $(u_0, u_1)$, a wrong candidate function $S_0^i$ would be got which is an affine equivalent to $S_0^i$, namely there exists an $8 \times 8$-bit matrix $a$ and an $8$-bit vector $b$ such that $S_0^i(\cdot) = a \cdot S_0^i(\cdot) + b$, with $a \neq 0$ and $(a, b) \neq (0, 1)$. The function $S^{-1} \circ S_0^i$ satisfies

$$S^{-1} \circ S_0^i = S^{-1} \circ (a \cdot S(rk_{i,0} \circ E^{-1}_{i+1,0}(\cdot)) + b).$$

In this case, $S^{-1} \circ S_0^i(\cdot)$ has an algebraic degree greater than 1 with an overwhelming probability. More specifically, we set the function $\hat{g}(\cdot) = S^{-1} \circ S_0^i(\cdot)$, used Lai’s higher-order derivative concept [21] to calculate the first-order derivative of $\hat{g}$, and finally ran ten thousand tests without obtaining a function with an algebraic degree of 1 or less. For instance, the first-order derivative $\hat{g}$ at point $(01)$ is set to $\hat{g}(x) = \hat{g}(x \oplus 01) \oplus \hat{g}(x)$, and we verify whether $\hat{g}(x)$ is constant with at most $2^7$ inputs of $x$, since $\hat{g}(x) = \hat{g}(x \oplus 01)$. For each wrong pair, the probability that $\hat{g}(x)$ is constant is roughly $2^{-8}$, so wrong guesses can be quickly removed.

After recovering $S_0^i$, we can use Eq. (8) to recover $S_1^i$ by exhaustive search on $v_0$, and similarly recover $S_2^i$ and $S_3^i$ with other equations finally.

3) Recovering the Linear Part of Encoding $E_{i+1}$: After the $S_j^i$ functions have been recovered ($j = 0, 1, 2, 3$), however it is not as easy to recover the output encodings $E_{i+1,0}$ as Lepoint et al.’s attack on Chow et al.’s white-box AES implementation, because of the existence of the unknown constant $\epsilon_i$, which is partially due to the different structures of Feistel and SPN ciphers and the design of Yao and Chen’s white-box SM4 implementation. Anyway, we find a trick to recover the linear part of the output encodings $E_{i+1,0}$.

Since $E_{i+1,0}$ is an invertible affine transformation, we write...
Given a 32-bit input $Z_i = (z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$ to the $f^i$ collision function, denote the original 32-bit value immediately after the L operation as follows:

$$W_i = \begin{bmatrix} W_{i,0} & W_{i,1} & W_{i,2} & W_{i,3} \end{bmatrix}^T = L_0 \circ S_j^0(z_{i,0}) \oplus L_1 \circ S_j^1(z_{i,1}) \oplus L_2 \circ S_j^2(z_{i,2}) \oplus L_3 \circ S_j^3(z_{i,3}).$$

As L is public and we have recovered $S_j^j$ above ($j = 0, 1, 2, 3$), we can compute $Y_j$. The output of the $f^i$ collision function is

$$f^i = \begin{bmatrix} f_{0}^i & f_{1}^i & f_{2}^i & f_{3}^i \end{bmatrix}^T = \begin{bmatrix} E_{i+1,0}(W_{i,0} \oplus \varepsilon_{i,0}) & E_{i+1,1}(W_{i,1} \oplus \varepsilon_{i,1}) & E_{i+1,2}(W_{i,2} \oplus \varepsilon_{i,2}) & E_{i+1,3}(W_{i,3} \oplus \varepsilon_{i,3}) \end{bmatrix},$$

where $(\varepsilon_{i,0}, \varepsilon_{i,1}, \varepsilon_{i,2}, \varepsilon_{i,3}) = \varepsilon_i$.

Subsequently, to recover $E_{i+1,j}$, we need to know the 8-bit unknown constant $\varepsilon_{i,j}$. A straightforward way is to try by exhaustive search, which would cause an additional complexity factor of $2^8$. However, we find we can recover the linear part $C_{i+1,j}$ at ease with a negligible time complexity, as follows.

First, we consider the output of the arbitrary 32-bit input $Z_i = (z_{i,0}, z_{i,1}, z_{i,2}, z_{i,3})$ under the $f_0^i$ collision function,

$$f_0^i(Z_i) = E_{i+1,0}(W_{i,0} \oplus \varepsilon_{i,0}) = C_{i+1,0}(W_{i,0} \oplus C_{i+1,0}(\varepsilon_{i,0}) \oplus c_{i+1,0}, \varepsilon_{i,0}).$$

where $W_{i,0}$ is defined above, which denotes the corresponding original 8-bit value immediately after the L operation under the input $X$.

Next, we choose the 32-bit input $Z_0 = (z_{0,0}, z_{0,1}, z_{0,2}, z_{0,3})$ to the $f^i$ collision function, so that the original 32-bit value immediately after the L operation is 0; this can be done easily by choosing $Z_0$ such that

$$(S_j^0(z_{0,0}), S_j^1(z_{0,1}), S_j^2(z_{0,2}), S_j^3(z_{0,3})) = L^{-1}(0) = 0.$$ Thus, its corresponding output under the $f_0^i$ collision function is

$$f_0^i(Z_0) = C_{i+1,0}(\varepsilon_{i,0}) \oplus c_{i+1,0}.$$ At last, XORing Eq. (9) and Eq. (10), we get $f_0^i(Z_i) \oplus f_0^i(Z_0) = C_{i+1,0}(W_{i,0})$. As a consequence, we can recover the linear part $C_{i+1,0}$ of the output encodings $E_{i+1,0}$. The linear parts of other output encodings $E_{i+1,j}$ can be recovered similarly.

4) Recovering the Masked Round Key $rk_{i+1} \oplus C_{i+1}^{-1}(c_{i+1})$.

We similarly define the collision function $f^{i+1}$ starting from the $(i+1)$-th round. Subsequently, as Lepoint et al. did on Chow et al.’s white-box AES implementation, we would target to recover the round key $rk_{i+1}$ in the following $(i+1)$-th round, but nevertheless we cannot recover a round key byte from the collision function, because $\varepsilon_{i+1}$ and $E_{i+1,j}$ are unknown, although the linear part $C_{i+1,j}$ can be recovered as above. At present we can only recover the masked round key $rk_{i+1} \oplus C_{i+1}^{-1}(c_{i+1})$, as follows.

Based on the equations similar to Eq. (4) and Eq. (5), we define

$$g(x) = f_0^{i+1}(E_{i+1,0}(S^{-1}(x) \oplus r_{k_{i+1},0}), 0, 0, 0)$$
$$= f_0^{i+1}(C_{i+1,0}(S^{-1}(x)) \oplus E_{i+1,0}(r_{k_{i+1},0}), 0, 0, 0)$$
$$= E_{i+2,0}(B_1(x) \oplus \delta \oplus \varepsilon_{i+1,0}),$$

where $\delta = B_2 \circ S_{i+1}^{-1}(0) \oplus B_2 \circ S_2^{i+1}(0) \oplus B_3 \circ S_3^{i+1}(0)$ is a constant that can be easily computed.

Because $E_{i+2,0}$ is an 8 $\times$ 8-bit affine transformation, the function $g$ has an algebraic degree of at most 1. For a wrong guess $E_{i+1,0}(r_{k_{i+1},0}) \neq E_{i+1,0}(r_{k_{i+1},0})$, the function $g$ is defined as

$$\hat{g}(x) = f_0^{i+1}(C_{i+1,0}(S^{-1}(x)) \oplus E_{i+1,0}(r_{k_{i+1},0}), 0, 0, 0)$$
$$= E_{i+2,0}(B_1 \circ S(S^{-1}(x) \oplus r_{k_{i+1},0} + r_{k_{i+1},0}) \oplus \delta \oplus \varepsilon_{i+1,0}).$$

In this case, with a similar test, $\hat{g}$ has an algebraic degree of more than 1 with an overwhelming probability. We extract $E_{i+1,0}(r_{k_{i+1},0})$ by exhaustive search, that is, similarly we verify whether the first-order derivative $\varphi(x) = \hat{g}(x \oplus 01) \oplus \hat{g}(x)$ of $\hat{g}(x)$ at point 01 is constant for each guess $E_{i+1,0}(r_{k_{i+1},0})$. For a wrong guess $E_{i+1,0}(r_{k_{i+1},0})$, the probability that $\varphi(x)$ is constant is roughly $2^{-8}$, so wrong guesses can be quickly removed.

As a result, we can also recover $E_{i+1,0}(r_{k_{i+1},0})$ for $i = 1, 2, 3$, by changing the definition of the function $g$. Since $E_{i+1}(r_{k_{i+1}}) = C_{i+1}(r_{k_{i+1}}) \oplus c_{i+1} = C_{i+1}(r_{k_{i+1}}) \oplus C_{i+1}^{-1}(c_{i+1})$, we can get the masked round key $r_{k_{i+1}} \oplus C_{i+1}^{-1}(c_{i+1})$, where $C_{i+1}^{-1}(c_{i+1})$ is an unknown Boolean mask.

5) Recovering Encodings $P_{i+4}$, $P_{i+4}'$ Partially or Completely: After recovering $S_0$, $S_1$, $S_2$ and $S_3$ functions in the first phase, we can easily recover the general $32 \times 32$-bit affine encoding $P_{i+4}$ used in the $i$-th round, see Fig. 2 or Fig. 4, since $Y_0'' = P_{i+4}^{20}(L \circ (S_0(z_{0,0})|S_1(z_{0,1})|S_2(z_{0,2})|S_3(z_{0,3}))).$

Even further, recall that $P_i(x) = A_i \cdot x \oplus a_i$, $P_{i+4}'(x) = P_{i+4} \oplus a_{i+4}$ and $P_{i+4}''(x) = P_{i+4} \oplus a_{i+4}$, we can learn that $a_i = a_{i+4} \oplus a_{i+4}'$, and consequently we can recover both the linear part $A_{i+4}$ and the constant part $a_{i+4}'$ of $P_{i+4}'$, as follows. First, we recover the linear part $A_{i+4}$ in a way similar to recovering the linear part of encoding $E_{i+1}$ above, that is, by considering the output difference under a pair of inputs:

$$Y_0'' \oplus Y_0'' = P_{i+4}''(L \circ (S_0(z_{0,0})|S_1(z_{0,1})|S_2(z_{0,2})|S_3(z_{0,3}))) \oplus P_{i+4}'(L \circ (S_0(z_{0,0})|S_1(z_{0,1})|S_2(z_{0,2})|S_3(z_{0,3}))) = A_{i+4}(L \circ (S_0(z_{0,0})|S_1(z_{0,1})|S_2(z_{0,2})|S_3(z_{0,3})) \oplus L \circ (S_0(z_{0,0})|S_1(z_{0,1})|S_2(z_{0,2})|S_3(z_{0,3}))).$$
6) Time Complexity: In the phase of recovering $S_0$, there are $2^{16}$ candidates $(u_0, u_1)$ for exhaustive search, and to verify whether $\varphi(x)$ is constant we need to calculate $\varphi(x)$ for at most $2^7$ inputs. For a wrong guess $(u_0, u_1)$, the probability that $\varphi(x)$ is constant is $2^{-5}$ roughly. Thus, the expected value of the test is $1 + 1/256 + \cdots + 1/(256^{127}) \approx 1.0$. The expected time complexity of recovering $S_0$ is hence about $2^{16} \cdot 1 \cdot 2 = 2^{17}$ (dominated by $S_i/S_i^{-1}$ computations). We recover $S_1$, $S_2$, and $S_3$ by exhaustive search on $v_0$ and produce an expected time complexity of $3 \cdot (2^8 \cdot 1 \cdot 2) = 3 \cdot 2^9$. Thus, the expected time complexity of recovering all the four $S_j$’s is about $2^{17} + 3 \cdot 2^9 = 259 \cdot 2^9$.

The time complexity for recovering the linear part of output encoding $E_{i+1,j}$ is negligible. The expected time complexity of recovering $E_{i+1,0}(r_k_{i+1,0})$ is about $2^8 \cdot 1 \cdot 2 = 2^9$, so the total expected time complexity of recovering $E_{i+1}(r_k_{i+1})$ is about $259 \cdot 2^9 + 4 \cdot (2^8 \cdot 1 \cdot 2) = 259 \cdot 2^9 + 2^{11}$, and the expected time complexity is about $31 \times (259 \cdot 2^9 + 2^{11}) \approx 2^{22}$ to recover all the masked round keys $r_k_{i+1} \oplus C_{i+1}^{-1}(c_{i+1})$ in the last 31 rounds ($i = 1, 2, \cdots, 31$).

7) A Somewhat Equivalent White-Box SM4 Implementation Mostly with Boolean Masks: After we recover both the linear and constant parts of encoding $P_{i+4}^P$ and the linear parts of encodings $P_{i+4}^I$ and $E_{i+1}$ above ($i = 0, 1, \cdots, 30$), we can peel off the affine encodings $P_{i+4}^P$ and the linear parts $A_{i+4}$ and $C_{i+1}$ of the affine encodings $P_{i+4}^I$ and $E_{i+1}$ in the 32 rounds, and thus it is easy to see that we can simplify Yao and Chen’s white-box SM4 implementation into such an equivalent white-box SM4 implementation that the 32 rounds have only Boolean masks for the encodings except $P_0$, $P_1$, $P_2$, $P_3$, $E_0$ and $P_{35}$; in particular, each $X_{i+4}$ is masked by an unknown Boolean constant, and the $(i+1)$-th round transformation $T$ has masked input but original output. Note that by “somewhat equivalent” we mean this simplifying process still has a workload, and thus the simplifying process is costly when one would like to have a security expectation below $2^{22}$, but it is desirable on the contrary, since the cumbersome affine encodings are not required to generate any longer from the security’s perspective.

So far we can only recover the masked round keys $r_k_{i+1} \oplus C_{i+1}^{-1}(c_{i+1})$, and it seems impossible to work out the original key further, and alternatively one may prove this is impossible further. We leave it as an open problem.

IV. CRYPTANALYSIS OF XIAO AND LAI’S WHITE-BOX SM4 IMPLEMENTATION

Xiao and Lai’s white-box SM4 implementation [38] is similar to Yao and Chen’s white-box SM4 implementation at a high level, except that there is no state expansion to the S-box layer and thus the original $L$ operation is used. Fig. 5 depicts an encryption round of Xiao and Lai’s white-box SM4 implementation, where $Q_i$ is a general invertible affine output encoding. Therefore, we can apply our above cryptanalysis to Xiao and Lai’s white-box SM4 implementation in the same way, and the total expected time complexity is also about $2^{22}$ for recovering recover all the masked round keys in the last 31 rounds. Likewise, we can peel off most white-box operations and make a somewhat equivalent white-box SM4 implementation mostly with Boolean masks.

V. CRYPTANALYSIS OF SHANG’S WHITE-BOX SM4 IMPLEMENTATION

Shang’s white-box SM4 implementation [31] is based on Xiao and Lai’s white-box SM4 implementation, mainly by applying two general 16-bit affine encodings $E_{i,0}$ and $E_{i,1}$ to the input of the S-box layer in parallel, each corresponding to two S-boxes and subsequently a $32 \times 16$-bit component $L_0$ or $L_1$ of the L matrix, and thus two $16 \times 32$-bit tables. Fig. 6 depicts an encryption round of Shang’s white-box SM4 implementation.

We can similarly exploit our above collision-based attack to Shang’s white-box SM4 implementation after a few modifications, to recover some masked round keys and peel off most white-box operations to a somewhat equivalent implementation. Specifically, we represent $L$ by Eq. (3) with two $16 \times 16$-bit blocks $L_0$ and $L_1$ as

$$L = \begin{bmatrix} L_0 & L_1 \\ L_1 & L_0 \end{bmatrix},$$

define

$$S_0(x) = \begin{bmatrix} S \\ S \end{bmatrix} \begin{bmatrix} (r_k_{i,0}) (r_k_{i,1}) \oplus E_{i,0}(x) \end{bmatrix},$$

$$S_1(x) = \begin{bmatrix} S \\ S \end{bmatrix} \begin{bmatrix} (r_k_{i,2}) (r_k_{i,3}) \oplus E_{i,1}(x) \end{bmatrix},$$

where $x \in GF(2)^{16}$, and define a collision function on Shang’s white-box SM4 implementation whose equivalent $f^i$
is depicted in Fig. 7, as follows:

\[ f'(z_{i,0}, z_{i,1}) = [f'_0(z_{i,0}, z_{i,1}), f'_1(z_{i,0}, z_{i,1})]^T \]

\[ = \begin{bmatrix} E^{-1}_{i+1,0} \\ E^{-1}_{i+1,1} \end{bmatrix} \oplus \epsilon_i \circ L_0 \circ \begin{bmatrix} \bar{S}_i \\ \bar{S}_i \end{bmatrix} \begin{bmatrix} (r_{k_i,0} | r_{k_i,1}) \oplus E_{i,0}(z_{i,0}) \\ (r_{k_i,2} | r_{k_i,3}) \oplus E_{i,1}(z_{i,1}) \end{bmatrix} \]

\[ = \begin{bmatrix} E^{-1}_{i+1,0} \\ E^{-1}_{i+1,1} \end{bmatrix} \oplus \epsilon_i \circ L_0 \circ \begin{bmatrix} S'_i(z_{i,0}) \\ S'_i(z_{i,1}) \end{bmatrix}. \]

Then, we consider the collision \( f'_i(\alpha, 0) = f'_i(0, \beta), \) where \( \alpha, \beta \in \text{GF}(2)^{32} \) and \( h = 0, 1. \) At last, we get the following linear system of \( 2 \times (2^{16} - 1) \) equations with \( 2 \times (2^{16} - 1) \) unknowns:

\[ \hat{L}_0 \circ u'_0 + \hat{L}_1 \circ v'_0 = 0, \]

\[ \hat{L}_1 \circ u'_0 + \hat{L}_0 \circ v'_0 = 0, \]

where \( u'_0 = S'_i(\alpha) \oplus S'_i(0) \) and \( v'_0 = S'_i(\beta) \oplus S'_i(0) \) and \( (\alpha, \beta) \neq (0, 0). \) Subsequently, by a similar process, we can recover the \( S'_i \) function with an expected time complexity of about \( 2^{32} \cdot 1 \cdot 2 \cdot 2 + 2^{16} \cdot 1 \cdot 2 \cdot 2 = 2^{34} + 2^{18} \) and recover the linear part of \( E_{i+1,0} \) with a negligible time complexity. Note that here each \( S'_i \) computation involves two \( S \) computations.

At last, suppose that \( E_{i+1,0}(\cdot) = C_{i+1,0}(\cdot) + c_{i+1,0} \) and the linear part \( C_{i+1,0} \) has been recovered as above, where \( C_{i+1,0} \) is a general invertible \( 16 \times 16 \)-bit matrix and \( c_{i+1,0} \) is a 1-bit constant. Similarly, we depend on the following function to recover the masked round key \( (r_{k_i+1,0} | r_{k_i+1,1}) + C_{i+1,0}(\oplus c_{i+1,0}) \) with an expected time complexity of about \( 2^{16} \cdot 2 \cdot 2 = 2^{18} \):

\[ f_{i+1}(E_{i+1,0}(0 | S^{-1}_1) + (x) \oplus (r_{k_i+1,0} | r_{k_i+1,1})), 0) \]

\[ = f_{i+1}(C_{i+1,0}(S^{-1}_1) + (x) \oplus E_{i+1,0}(r_{k_i+1,0} | r_{k_i+1,1}), 0) \]

\[ = E^{-1}_{i+2,0}(L_0(x) + L_1 \circ S^{-1}_{i+1}(0) \oplus \varepsilon_i+1,0), \]

where \( \varepsilon_{i+1,0} \) is the corresponding 16-bit part of \( \varepsilon_{i+1}. \)

Thus, the total expected time complexity is about \( 2^{34} \cdot 2^{18} + 2 \cdot 2^{18} \approx 2^{34} \) for recovering the masked round key \( r_{k_i+1} \oplus C_{i+1,0}(\oplus c_{i+1,0}) \) from Shang’s white-box SM4 implementation, and the expected time complexity is about \( 31 \times 2^{34} \approx 2^{39} \) to recover all the masked round keys \( r_{k_i+1} \oplus C_{i+1,0}(\oplus c_{i+1,0}) \) in the last 31 rounds \((i = 1, 2, \cdots, 31). \) As a consequence, we can similarly peel off most white-box operations and make a somewhat equivalent white-box SM4 implementation mostly with Boolean masks.

VI. Collision-Based Attack on Wu et al.’s White-Box SM4 Implementation

In this section, we briefly describe Wu et al.’s white-box SM4 implementation and our attack.

A. Wu et al.’s White-Box SM4 Implementation

An encryption round of Wu et al.’s white-box SM4 implementation [35] is made up of three parts, as depicted in Fig. 8, but it is expanded to 36 rounds to produce the original output (without protection), and there are respectively two types of lookup tables in the second and third parts, especially, the second type of lookup tables uses three different construction methods for different rounds.

1) Part 1: The first part is processed as follows, and the 32-bit output \( Z_i \) is protected by a diagonal invertible matrix \( E_i \):

\[ X'_{i,j} = A_{i,j}(X_{i+1}) \cdot 1, 2, 3; \]

\[ Z_i = X'_{i+1} + X'_{i+2} + X'_{i+3}, \]

where \( A_{i,j} \) is a composite 32-bit invertible matrix with different encodings \((i = 0, 1, \ldots, 35)\), as follows,

\[ \begin{align*}
A_{0,0} &= E_0 P, & A_{0,2} &= E_0 P, & A_{0,3} &= E_0 P, \\
A_{1,1} &= E_1 P, & A_{1,2} &= E_1 P, & A_{1,3} &= E_1 R_{0}^{-1}, \\
A_{2,1} &= E_2 P, & A_{2,2} &= E_2 R_{0}^{-1}, & A_{2,3} &= E_2 R_{1}^{-1}, \\
A_{3,1} &= E_3 R_{0}^{-1}, & A_{3,2} &= E_3 R_{1}^{-1}, & A_{3,3} &= E_3 R_{2}^{-1}, \\
A_{4,1} &= E_4 R_{1}^{-1}, & A_{4,2} &= E_4 R_{2}^{-1}, & A_{4,3} &= E_4 R_{3}^{-1}, \\
\vdots & & & & & \vdots \\
A_{34,1} &= E_{34} R_{28}^{-1}, & A_{34,2} &= E_{34} R_{28}, & A_{34,3} &= E_{34} R_{29}, \\
A_{35,1} &= E_{35} R_{28}, & A_{35,2} &= E_{35} R_{29}, & A_{35,3} &= E_{35} R_{30}, \\
\end{align*} \]

with \( P = \text{diag}(P_0, P_1, P_2, P_3) \) being a diagonal invertible matrix and \( R_0, R_1, \ldots, R_30 \) being general invertible matrices.

2) Part 2: Construct four 8-bit to 32-bit lookup tables of the first type \( Table_{i,j} (j = 0, 1, 2, 3) \), and XOR the outputs of the four tables into a 32-bit \( Y_i \), as follows:

\[ Y_i = \bigoplus_{j=0}^{3} Table_{i,j} \]

\[ = Q_i \circ P \circ L \circ \begin{bmatrix} S \circ \oplus r_{k_i,0} \circ P_{0}^{-1} \circ E_{i,0}^{-1}(z_{i,0}) \\ S \circ \oplus r_{k_i,1} \circ P_{1}^{-1} \circ E_{i,1}(z_{i,1}) \\ S \circ \oplus r_{k_i,2} \circ P_{2}^{-1} \circ E_{i,2}^{+1}(z_{i,2}) \\ S \circ \oplus r_{k_i,3} \circ P_{3}^{-1} \circ E_{i,3}^{+1}(z_{i,3}) \end{bmatrix} \]

Figure 8. An encryption round of Wu et al.’s white-box SM4 implementation
3) Part 3: Construct four 16-bit to 8-bit lookup tables of the second type. Let \((X_i, Y_i)\) be the input of the four tables, and \(X_{i+4}\) be the output, then

- **Rounds 1-4:**
  \[X_i' = R_i \circ P(X_i), Y_i' = R_i \circ Q_i^{-1}(Y_i),\]
  \[X_{i+4} = Y_i' \oplus X_i';\]

- **Rounds 5-32:**
  \[X_i' = R_i \circ R_{i-4}(X_i), Y_i' = R_i \circ Q_i^{-1}(Y_i),\]
  \[X_{i+4} = Y_i' \oplus X_i';\]

- **Rounds 33-36:**
  \[X_i' = R_i \circ R_{i-8}(X_i), Y_i' = P^{-1} \circ Q_i^{-1}(Y_i),\]
  \[X_{i+4} = Y_i' \oplus X_i'.\]

B. Attacking Wu et al.’s Implementation

Note that all the encodings are linear (invertible matrices) in Wu et al.’s white-box SM4 implementation, rather than affine encodings as in the above three white-box SM4 implementations. As a consequence, we can devise a collision function in a similar way as above, but much easier, since there is no effect of unknown constant parts associated with the encodings. Fig. 9 (top) depicts a collision function on Wu et al.’s white-box SM4 implementation, where \(Q_i\) and \(Q_i^{-1}\) are cancelled with each other.

Further, the collision function can be simplified. First, we set \(X_i = 0\), and thus \(R_i \circ P(X_i) = 0\), since \(R_i\) and \(P\) are invertible matrices. Second, \(R_i\) and \(R_i^{-1}\) are cancelled with each other. Thus, after we adjust the position of encoding \(P\), we get a simplified collision function as depicted in Fig. 9 (bottom), that is

\[
f(z_i,0,z_{i,1}, z_{i,2}, z_{i,3}) = \begin{bmatrix} f_0(z_i,0, z_{i,1}, z_{i,2}, z_{i,3}), & f_1(z_i,0, z_{i,1}, z_{i,2}, z_{i,3}), & f_2(z_i,0, z_{i,1}, z_{i,2}, z_{i,3}), & f_3(z_i,0, z_{i,1}, z_{i,2}, z_{i,3}) \end{bmatrix}^T
\]

\[
= \begin{bmatrix} \hat{E}_{i+1,0}^{-1} & \hat{E}_{i+1,1} & \hat{E}_{i+1,2} & \hat{E}_{i+1,3}^{-1} \end{bmatrix} \circ \begin{bmatrix} S \circ \oplus_{r_k} \circ \hat{E}_{i+1,0}^{-1}(z_i) \\ S \circ \oplus_{r_k} \circ \hat{E}_{i+1,1}(z_1) \\ S \circ \oplus_{r_k} \circ \hat{E}_{i+1,2}(z_2) \\ S \circ \oplus_{r_k} \circ \hat{E}_{i+1,3}^{-1}(z_3) \end{bmatrix}
\]

where \(\hat{E}_{i+1,0}^{-1}(\cdot) = P_i^{-1} \circ E_{i+1,0}(\cdot)\) and \(\hat{E}_{i+1,1}(\cdot) = E_{i+1,1} \circ P_i(\cdot)\) are invertible 8 \times 8-bit matrices (\(j = 0, 1, 2, 3\)). Note that \(E_{i+1,j}\) and \(\hat{E}_{i+1,j}\) are matrices simply, not affine transformations, and thus this attack is a simplified case.

Define \(S_j\) function as

\[
S_j(\cdot) = S \circ \oplus_{r_k} \circ E_{i+1,j}^{-1}(\cdot) = S(r_k \circ E_{i+1,j}^{-1}(\cdot)), \quad j = 0, 1, 2, 3.
\]

Then, we can similarly recover the four functions \(S_j\)'s with an expected time complexity of about \(2^{16} \cdot 1 \cdot 2^3 \cdot (2^8 \cdot 1 \cdot 2) = 259 \cdot 2^9\). Since \(\hat{E}_{i+1,0}\) is an invertible 8 \times 8-bit matrix, we can recover it immediately by calculating \(\hat{E}_{i+1,0}(\cdot) = f_0(\psi^{-1}(\cdot), 0, 0, 0)\), where \(\psi : 0 \rightarrow B_1 \circ S_0(\alpha) \oplus B_2 \circ S_2(0) \oplus B_3 \circ S_3(0)\). Similarly for recovering \(\hat{E}_{i+1,1}, \hat{E}_{i+1,2}\) and \(\hat{E}_{i+1,3}\).

\[\begin{array}{c}
\frac{P}{R_i} \\
\frac{R_i}{L_i} \\
\frac{P}{R_i+4} \\
\frac{P}{R_i} \\
\frac{P}{R_i+4} \\
\end{array}\]

\[\begin{array}{c}
\frac{P_i}{P_i+1} \\
\frac{P_i}{P_i+1} \\
\frac{P_i}{P_i+1} \\
\frac{P_i}{P_i+1} \\
\end{array}\]

Figure 9. Collision function and its equivalent on Wu et al.’s white-box SM4 implementation

At last, we set function \(g\) as

\[g(x) = f_j(\hat{E}_{i+1,0}(S^{-1}(x) \oplus r_k), 0, 0, 0) = \hat{E}_{i+1,0}(B_1 \circ x \oplus B_2 \circ S_1(0) \oplus B_2 \circ S_2(0) \oplus B_3 \circ S_3(0)),\]

and we can similarly recover the round key byte \(r_k\) and finally the whole round key \(r_k\) with an expected time complexity of about \(4 \cdot (2^8 \cdot 1 \cdot 2) = 2^{11}\). Therefore, the total expected time complexity is about \(259 \cdot 2^9 + 2^{11} \approx 2^{17.1}\) for recovering a round key.

At last, note that producing unprotected ciphertexts would make Wu et al.’s white-box SM4 implementation be potentially vulnerable to other attacks, see [8].

VII. CONCLUDING REMARKS

The SM4 block cipher is a Chinese national standard and an ISO international standard, formerly known as SMS4, and a few white-box SM4 implementations have been proposed with its increasingly wide use. In this paper, we have analysed the security of a type of white-box SM4 constructions that use an affine diagonal block encoding to protect the original XOR sum of the three branches entering the S-box layer of a round and use its inverse to protect the original input of the S-box layer, and have shown that it can be somewhat equivalent to a plain implementation mostly with Boolean masks for the affine case and is insecure for the linear case, by devising collision-based attacks to peel off most white-box operations and recover a masked or original round key. Our cryptanalysis results show that generating such a white-box SM4 implementation with affine encodings can be simplified into generating a plain implementation with Boolean masks (if its security expectation is beyond the given complexity), avoiding the cumbersome affine encodings, and the effect of an affine encoding is significantly better than the effect of a linear encoding in the sense of security. Currently we think it seems impossible to recover the original key from the masked round keys, and leave it as an open problem.
ACKNOWLEDGMENTS

This is an extended version of the paper appeared in Proceedings of ISC 2021 — The 24th Information Security Conference [26]. In this extended version, we revised and corrected our previous cryptanalysis results and conclusions on Yao and Chen’s and Xiao and Lai’s white-box SM4 implementations, and gave cryptanalysis results on two other white-box SM4 implementations, namely Shang’s and Wu et al.’s implementations. The authors are very grateful to Prof. Zheng Gong for pointing out a mistake in an earlier version of this paper. Jiqiang Lu is Qianjiang Special Expert of Hangzhou.

REFERENCES