SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2018, Vol. 61 032113:1–032113:18
https://doi.org/10.1007/s11432-017-9230-8

A real-time inversion attack on the GMR-2 cipher used in the satellite phones

Jiao HU, Ruilin LI* & Chaojing TANG

College of Electronic Science, National University of Defense Technology, Changsha 410073*, China*

Received 12 May 2017/Accepted 19 July 2017/Published online 1 February 2018

Abstract The GMR-2 cipher is a type of stream cipher currently being used in some inmarsat satellite phones. It has been proven that such a cipher can be cracked using only one single-frame (15 bytes) known keystream but with moderate executing time. In this paper, we present a new thorough security analysis of the GMR-2 cipher. We first study the inverse properties of the cipher's components to reveal a bad one-way character of the cipher. By then introducing a new concept called "valid key chain" according to the cipher's key schedule, we propose an unprecedented real-time inversion attack using a single-frame keystream. This attack comprises three phases: (1) table generation; (2) dynamic table look-up, filtration and combination; and (3) verification. Our analysis shows that, using the proposed attack, the size of the exhaustive search space for the 64-bit encryption key can be reduced to approximately 2^{13} when a single-frame keystream is available. Compared with previous known attacks, this inversion attack is much more efficient. Finally, the proposed attack is carried out on a 3.3-GHz PC, and the experimental results thus obtained demonstrate that the 64-bit encryption-key could be recovered in approximately 0.02 s on average.

Keywords satellite phone, stream cipher, GMR-2, cryptanalysis, inversion attack

Citation Hu J, Li R L, Tang C J. A real-time inversion attack on the GMR-2 cipher used in the satellite phones. Sci China Inf Sci, 2018, 61(3): 032113, https://doi.org/10.1007/s11432-017-9230-8

1 Introduction

1.1 Backgrounds and the GMR-2 cipher

With the rapid evolution and development of 4G technologies, nowadays, mobile phone systems are available worldwide; however, it is difficult to build a complete mobile network in some remote areas such as outlying desert areas, oceans, and mountains. Thus, to fill the gaps left behind by radio-based technologies, the use of satellite phones has become widespread in these areas. Currently, the commonly used satellite communication standards are primarily developed by international standards organization ETSI [1]; this includes the GMR-1 and the GMR-2 standards. For instance, Thuraya phones are based on the GMR-1 standard, while the Inmarsat phones adopt GMR-2 standard.

Given that the confidentiality is a very crucial aspect in satellite communications, the encryption algorithms in the satellite phones should be strong enough to withstand various eavesdropping risks. In the mobile application scenario, several symmetric ciphers have been developed and adopted as cryptographic components for secure communications, e.g., A5, SNOW, and ZUC, and their security has been sufficiently evaluated in the past years [2–8]. However, the GMR cryptographic algorithms are not included in the

* Corresponding author (email: securitylrl@gmail.com)

officially published GMR standards, and the details of these satellite cipher algorithms were non-public until the German research team Driessen et al. [9,10] uncovered the GMR-1 and the GMR-2 ciphers using reverse engineering in 2012. Their analysis results illustrate that both the aforementioned ciphers are stream ciphers. In particular, the GMR-1 cipher is a proprietary variant of the GSM A5/2 algorithm [9], and thus, the cryptanalytic methods against the A5/2 algorithm [11,12] can almost be well-adopted to it. The GMR-2 cipher is an entirely newly designed stream cipher; however, it has been found to be insecure in the case of two types of known plaintext attacks. Driessen et al. [9] proposed a known plaintext attack against it for the first time based on the read-collision technique according to the key-scheduling features of the GMR-2 cipher. This type of attack can reduce the size of the brute-force space from 2^{64} to approximately 2^{18} with approximately 50–65 bytes of the keystream. Li et al. [13] further put forward a low data complexity attack method called the dynamic guess-and-determine attack which can break the GMR-2 cipher by guessing approximately 28 bits on average when 15 bytes of the keystream are available.

1.2 Main contribution and the outline

Generally speaking, stream ciphers first generate keystreams by implementing a series of complex cryptographic transformation on the initial vectors and the encryption-key, and then perform an XOR operation on the keystreams with plaintexts in order to obtain the ciphertexts. Therefore, in order to resist a known plaintext attack, a vital requirement of stream ciphers is the one-way property, i.e., it must be difficult for the adversary to derive the encryption-key from the keystream using an inversion procedure. According to [14–16], Golic et al. proposed an inversion attack method against the keystream generator consisting of a linear feedback shift register and a nonlinear filter and proved the effectiveness of such an attack for some cases.

In this paper, we study the inverse properties of the GMR-2 cipher to reveal a bad one-way character of such a cipher. By then introducing a new concept "valid key chain", we propose what we call the inversion attack against the GMR-2 cipher using a single-frame (15 bytes) keystream. It should be noted that Ref. [13] also presents a low-data-complexity attack on the cipher. Such an attack is based on the inner structure of the cipher in the forward direction and adopts the dynamic guess-and-determine strategy, which finally reduces the size of the brute-force space to approximately 2^{28}. Our proposed attack mainly concentrates on the backward direction of the cipher and comprises three major phases: (1) table generation; (2) dynamic table look-up, filtration and combination; and (3) verification. With the help of an extra 6-KB memory storage, this inversion attack can reduce the size of the exhaustive search space from 2^{64} to approximately 2^{13} on average when a single-frame keystream is available. This indicates that the inversion attack is very efficient and practical, which could lead to a real-time crack on the GMR-2 cipher. The experimental results obtained with a 3.3-GHz PC demonstrate that the 64-bit encryption key can be completely retrieved in approximately 0.02 s.

This paper is organized as follows: a brief introduction to the GMR-2 cipher is presented in Section 2. Sections 3 and 4 present an analysis of the inverse properties of the three components of the GMR-2 cipher as well as the cipher itself. Section 5 presents the proposed attack strategy and details the attack procedure. The experimental results and the attack complexity are subsequently analyzed in Section 6. Finally, Section 7 presents a concise summary of this study.

2 Description of the GMR-2 cipher

The GMR-2 cipher is a type of stream cipher with 64-bit key. As shown in Figure 1, the internal states of the cipher include a 8-byte shift register $S = (S_7, S_6, \ldots, S_0)$, an 8-byte encryption-key register $K = (K_7, K_6, \ldots, K_0)$, a counter $c \in \{0, 1, \ldots, 7\}$, and a toggle-bit $t \in \{0, 1\}$. They are transformed through three components $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$. At each clock l, the cipher generates 1-byte keystream denoted by Z_l.

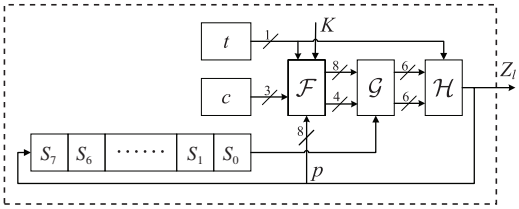

Figure 1 Overall structure of the GMR-2 cipher.

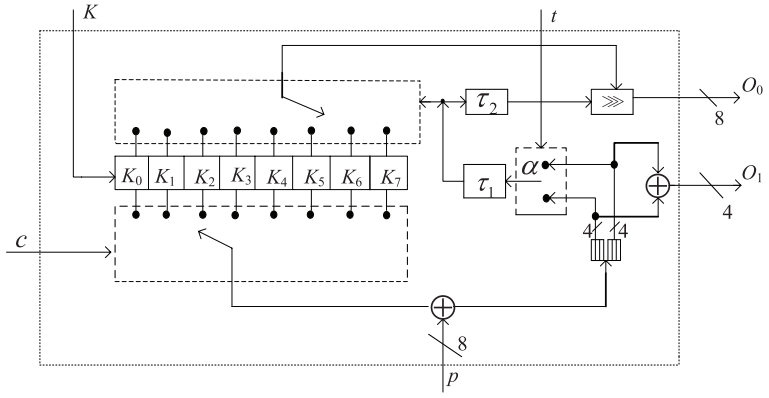

Figure 2 Structure of the $\mathcal{F}$-component.

Table 1 Definition of τ_1 and τ_2

α	$\tau_1(\alpha)$	$\tau_2(\tau_1(\alpha))$	α	$\tau_1(\alpha)$	$\tau_2(\tau_1(\alpha))$
(0,0,0,0)	2	6	(1,0,0,0)	3	7
(0,0,0,1)	5	3	(1,0,0,1)	0	4
(0,0,1,0)	0	4	(1,0,1,0)	6	2
(0,0,1,1)	6	2	(1,0,1,1)	1	5
(0,1,0,0)	3	7	(1,1,0,0)	5	3
(0,1,0,1)	7	1	(1,1,0,1)	7	1
(0,1,1,0)	4	4	(1,1,1,0)	4	4
(0,1,1,1)	1	5	(1,1,1,1)	2	6

The $\mathcal{F}$-component can be treated as a key schedule part of the GMR-2 cipher, it combines two bytes of the encryption-key with the previous output (a keystream byte) to compute a 12-bit output. The $\mathcal{G}$-component is designed for mixing purpose and is a linear function with a 12-bit input and 12-bit output. The $\mathcal{H}$-component is a nonlinear filter and comprises two parallel DES S-boxes with a 12-bit input and 8-bit output. Subsection 2.1 describes these three components in detail and Subsections 2.2 describes the mode operation of the cipher.

2.1 Components of the GMR-2 cipher

2.1.1 *$\mathcal{F}$-component*

As the most interesting part of the cipher, the internal structure of the $\mathcal{F}$-component is depicted in Figure 2. The 8-byte encryption key $K = (K_7, K_6, \ldots, K_0)$ is fed into a 64-bit register, and it remains unchanged during the execution of the entire cipher. At each clock, the $\mathcal{F}$-component selects two key bytes (one from the lower side and another from the upper side) for further computation, and the procedure can be formally described as follows.

Let us assume that the cipher is executed at the l-th clock. Besides the 8-byte encryption key K, the inputs of the $\mathcal{F}$-component also contain three variables c, t, and p, where $c = l \bmod 8$ is a counter ranging from 0 to 7 sequentially and repeatedly, $t = c \bmod 2$ is a toggle bit, and $p = (p_7, p_6, \ldots, p_0) \in \{0,1\}^8$ is a feedback keystream byte that has already been generated in the last clock. We simply use $p = Z_{l-1}$ to

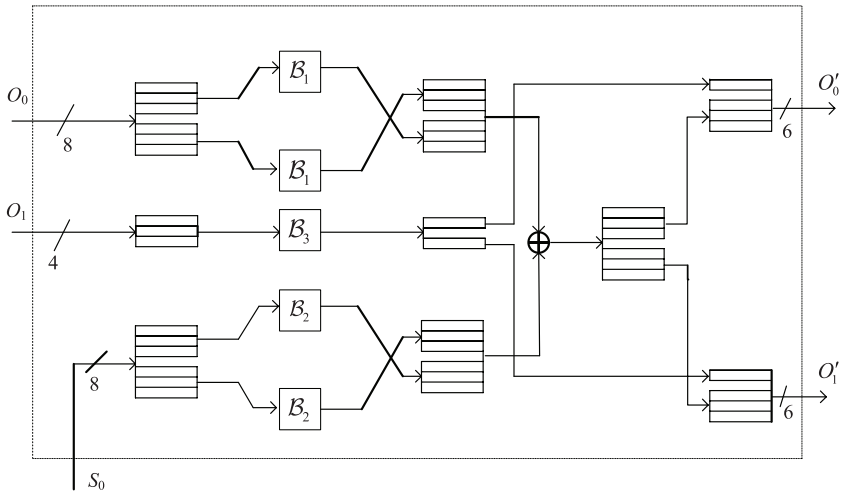

Figure 3 Structure of the $\mathcal{G}$-component.

denote the keystream byte that was generated at the previous (the $(l-1)$-th) clock. The outputs of the $\mathcal{F}$-component comprise an 8-bit O_0 and a 4-bit O_1 with the following definitions:

$$\begin{cases} O_0 = (K_{\tau_1(\alpha)} \ggg \tau_2(\tau_1(\alpha)))_8, \\ O_1 = ((((K_c \oplus p) \gg 4)\&\mathsf{0x0F}) \oplus ((K_c \oplus p)\&\mathsf{0x0F}))_4, \end{cases} \tag{1}$$

where α is defined by

$$\alpha = \mathcal{N}(t, K_c \oplus p) = \begin{cases} ((K_c \oplus p)\&\mathsf{0x0F})_4, & \text{if} \quad t = 0, \\ (((K_c \oplus p) \gg 4)\&\mathsf{0x0F})_4, & \text{if} \quad t = 1, \end{cases} \tag{2}$$

and $\tau_1 : \{0,1\}^4 \to \{0,1\}^3$, $\tau_2 : \{0,1\}^3 \to \{0,1\}^3$ are two functions implemented via table look-up as shown in Table 1.

2.1.2 *$\mathcal{G}$-component*

Figure 3 illustrates the structure of the $\mathcal{G}$-component, where $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ are all linear functions that return a 4-bit output on being provided a 4-bit input and have the following definitions:

$$\begin{cases} \mathcal{B}_1 : (x_3, x_2, x_1, x_0) \mapsto (x_3 \oplus x_0, x_3 \oplus x_2 \oplus x_0, x_3, x_1), \\ \mathcal{B}_2 : (x_3, x_2, x_1, x_0) \mapsto (x_1, x_3, x_0, x_2), \\ \mathcal{B}_3 : (x_3, x_2, x_1, x_0) \mapsto (x_2, x_0, x_3 \oplus x_1 \oplus x_0, x_3 \oplus x_0). \end{cases}$$

The $\mathcal{G}$-component obtains the outputs of the $\mathcal{F}$-component (O_0 and O_1) and 1-byte state register $S_0 = (S_{0,7}, S_{0,6}, \ldots, S_{0,0})$ as its inputs, and after a series of linear transformation, transposition, and XOR operations, it outputs two 6-bit outputs O'_0 and O'_1, which can be expressed as follows:

$$\begin{cases} O'_0 = (O_{0,7} \oplus O_{0,4} \oplus S_{0,5}, O_{0,7} \oplus O_{0,6} \oplus O_{0,4} \oplus S_{0,7}, O_{0,7} \oplus S_{0,4}, \\ \qquad O_{0,5} \oplus S_{0,6}, O_{1,3} \oplus O_{1,1} \oplus O_{1,0}, O_{1,3} \oplus O_{1,0})_6, \\ O'_1 = (O_{0,3} \oplus O_{0,0} \oplus S_{0,1}, O_{0,3} \oplus O_{0,2} \oplus O_{0,0} \oplus S_{0,3}, O_{0,3} \oplus S_{0,0}, \\ \qquad O_{0,1} \oplus S_{0,2}, O_{1,2}, O_{1,0})_6. \end{cases} \tag{3}$$

2.1.3 *$\mathcal{H}$-component*

The $\mathcal{H}$-component obtains the two 6-bit outputs of the $\mathcal{G}$-component as its input. Figure 4 shows the structure of $\mathcal{H}$-component which comprises two 6-in and 4-out S-boxes ($\mathbb{S}_2$ and $\mathbb{S}_6$) that are used in the DES algorithm (refer to Tables 2 and 3). However, these two S-boxes have been reordered to account for

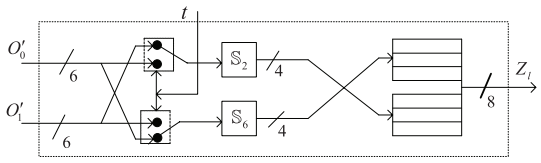

Figure 4 Structure of the $\mathcal{H}$-component.

Table 2 S-box $\mathbb{S}_2$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Table 3 S-box $\mathbb{S}_6$

	0	1	2	3	**4**	5	6	7	8	9	10	11	12	13	14	15
0	12	1	10	15	**9**	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	**9**	5	15	10	11	14	1	7	6	0	8	13

the different addressing. Let us assume that the 6-bit input of the S-box is $(x_5, x_4, x_3, x_2, x_1, x_0)_2$. For the GMR-2 cipher, the most-significant 4-bits (x_5, x_4, x_3, x_2) determine the column index of the S-box while the least-significant 2-bits (x_1, x_0) determine the S-box row index. Finally, depending on the toggle-bit t, the output 1-byte keystream can be defined as

$$Z_l = \begin{cases} (\mathbb{S}_2(O'_1), \mathbb{S}_6(O'_0))_8, & \text{if} \quad t = 0, \\ (\mathbb{S}_2(O'_0), \mathbb{S}_6(O'_1))_8, & \text{if} \quad t = 1. \end{cases} \tag{4}$$

2.2 Mode of operation

As mentioned in [9], we can now describe the mode of operation for the GMR-2 cipher. When the cipher is clocked at the l-th time, the state of the GMR-2 cipher will be changed as follows:

- The cipher generates 1-byte keystream Z_l based on the current value of the state-register S, the counter c and the toggle bit $t = c \bmod 2$.
- The counter c is incremented by 1, and when it attains a value of 8, it is reset to 0.
- The state-register S is shifted by 1 byte to the right, thus S_i=S_{i+1}, for $i = 0, 1, 2, \ldots, 6$, and S_7=Z_l. Meanwhile, $p = Z_l$[1)] is passed to the $\mathcal{F}$-component as the input parameter for the next iteration (the $(l+1)$-th clock).

The GMR-2 cipher is operated in two modes: the initialization mode and the generation mode.

Initialization mode. In the initialization phase, the following steps are performed:

- The counter $c = 0$ and the toggle-bit $t = 0$.
- The 64-bit encryption key is written into the register in the $\mathcal{F}$-component.
- The state register S is initialized with a 22-bit frame-number N according to a special rule, which is not detailed here as it is irrelevant to our attack.
- After c, t, S have been initialized, the cipher is clocked eight times, but the resulting keystream is discarded.

Generation mode. After the initialization is finished, the cipher is switched into generation mode in order to produce and output actual keystream bytes. We use $Z_l^{(N)}$ to denote the l-th keystream byte after initialization with the frame number N. For each frame number N, the cipher operates 15 clocks

1) When $l = 0$, the value of p is set to zero.

and generate a 15-byte keystream. After that, the frame number N automatically increases by 1 and the state register is re-initialized with the new frame number. The cipher then generates another 15-byte keystream. Based on the assumption that the frame numbering starts from 0, the actual keystream Z' is made up of blocks of 15 bytes that are concatenated as follows:

$$Z' = \left(Z_0^{(0)}, Z_1^{(0)}, \ldots, Z_{14}^{(0)}; Z_0^{(1)}, Z_1^{(1)}, \ldots, Z_{14}^{(1)}; Z_0^{(2)}, \ldots\right). \tag{5}$$

3 Inverse properties of the GMR-2 cipher's components

The GMR-2 cipher consists of three components, in which the $\mathcal{F}$-component plays the role of the key schedule, the $\mathcal{G}$-component acts as a linear transformation, and the $\mathcal{H}$-component implements a nonlinear transformation. Both the cryptanalytic methods proposed in [9,13] concentrate on the forward analysis of the GMR-2 cipher, whereas our proposed inversion attack focuses on the backward analysis, i.e., we attempt to reverse the encryption procedure to deduce the encryption key from the output keystream directly. Thus, in this section, we first study the inverse properties of the three components that are related to our subsequent analysis.

3.1 Inverse property of the $\mathcal{H}$-component

The $\mathcal{H}$-component comprises two parallel S-boxes, and it selects the column and row indices of the two S-boxes through the toggle-bit t, as show in Eq. (4). Furthermore, We can extract the relationship between the input (O'_0, O'_1) of $\mathcal{H}$ and the output Z_l of $\mathcal{H}$ (the keystream byte) by "inverting" the two S-boxes. Thus, we make the following proposition.

Proposition 1. For the $\mathcal{H}$-component, if the row index and the output of an S-box are known, then its column index can be uniquely determined. If only the outputs of both S-boxes are known, there will be $4 \times 4 = 16$ different corresponding row and column indices.

3.2 Inverse property of the $\mathcal{G}$-component

Let us assume that the shift register S_0 is known. The $\mathcal{G}$-component can then be represented by an affine transformation. We focus on how to extract the inputs O_0 and O_1 of $\mathcal{G}$, given the output O'_0 and O'_1 along with S_0. According to [13] and Eq. (3), the link[2)] between the input and output of the $\mathcal{G}$-component can be expressed as

$$\begin{cases} \boldsymbol{y}_1 = \boldsymbol{W}_1 \cdot \boldsymbol{x}_1 \oplus \boldsymbol{Q}' \cdot \boldsymbol{v}, \\ \boldsymbol{y}_2 = \boldsymbol{W}_2 \cdot \boldsymbol{x}_2, \end{cases} \tag{6}$$

where

$$\boldsymbol{W}_1 = \begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A} \end{pmatrix}, \quad \boldsymbol{W}_2 = (\boldsymbol{B}), \quad \boldsymbol{Q}' = \begin{pmatrix} \boldsymbol{C} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C} \end{pmatrix},$$

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{C} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad \boldsymbol{0} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

and

$$\begin{cases} \boldsymbol{y}_1 = \left(O'_{0,5}, O'_{0,4}, O'_{0,3}, O'_{0,2}, O'_{1,5}, O'_{1,4}, O'_{1,3}, O'_{1,2}\right)^{\mathrm{T}}, \\ \boldsymbol{y}_2 = \left(O'_{0,1}, O'_{0,0}, O'_{1,1}, O'_{1,0}\right)^{\mathrm{T}}, \end{cases}$$

2) It should be noted that the definition of the variable $\boldsymbol{v}$ in Eq. (6) in this paper is different from that in [13]. In fact, $\boldsymbol{Q}' \cdot \boldsymbol{v}$ in this paper is equivalent to $\boldsymbol{v}_1$ as defined in [13].

$$\begin{cases} \boldsymbol{x}_1 = (O_{0,7}, O_{0,6}, O_{0,5}, O_{0,4}, O_{0,3}, O_{0,2}, O_{0,1}, O_{0,0})^{\mathrm{T}}, \\ \boldsymbol{x}_2 = (O_{1,3}, O_{1,2}, O_{1,1}, O_{1,0})^{\mathrm{T}}, \\ \boldsymbol{v} = (S_{0,7}, S_{0,6}, S_{0,5}, S_{0,4}, S_{0,3}, S_{0,2}, S_{0,1}, S_{0,0})^{\mathrm{T}}. \end{cases}$$

In the above formulas, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{v}$ are used to represent O_0, O_1 and S_0, which is the input of $\mathcal{G}$, and $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ is used to represent a simple permutation of (O'_0, O'_1) which is the output of $\mathcal{G}$. On carefully observing the $\mathcal{H}$-component, it can be seen that $\boldsymbol{y}_1$ corresponds to the column indices of the two S-boxes, and $\boldsymbol{y}_2$ corresponds to the row indices of the two S-boxes.

Now if we treat $\boldsymbol{y}_1$, $\boldsymbol{y}_2$ and $\boldsymbol{v}$ (thus O'_0, O'_1, and S_0) as known values and $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ (thus O_0 and O_1) as unknown variables, the first and second formulas in Eq. (6) can be regarded as a system of linear equations with 8 and 4 variables, respectively. As both $\boldsymbol{A}$ and $\boldsymbol{B}$ are invertible matrices, we obtain

$$\begin{cases} \boldsymbol{x}_1 = \boldsymbol{W}_1^{-1} \cdot \boldsymbol{y}_1 \oplus \boldsymbol{Q} \cdot \boldsymbol{v}, \\ \boldsymbol{x}_2 = \boldsymbol{W}_2^{-1} \cdot \boldsymbol{y}_2, \end{cases} \tag{7}$$

where

$$\boldsymbol{W}_1^{-1} = \begin{pmatrix} \boldsymbol{A}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}^{-1} \end{pmatrix}, \quad \boldsymbol{W}_2^{-1} = (\boldsymbol{B}^{-1}), \quad \boldsymbol{Q} = \boldsymbol{W}_1^{-1} \cdot \boldsymbol{Q}' = \begin{pmatrix} \boldsymbol{A}^{-1} \cdot \boldsymbol{C} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}^{-1} \cdot \boldsymbol{C} \end{pmatrix},$$

$$\boldsymbol{A}^{-1} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}, \quad \boldsymbol{B}^{-1} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{A}^{-1} \cdot \boldsymbol{C} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

Therefore, we have the following proposition.

Proposition 2. For the $\mathcal{G}$-component, if O'_0, O'_1 and S_0 (thus $\boldsymbol{y}_1$, $\boldsymbol{y}_2$ and $\boldsymbol{v}$) are known values, then O_0 and O_1 (thus $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$) can be calculated directly using Eq. (7). Furthermore, O_0 (thus $\boldsymbol{x}_1$) is uniquely determined by $\boldsymbol{y}_1$, and O_1 (thus $\boldsymbol{x}_2$) is uniquely determined by $\boldsymbol{y}_2$.

3.3 Inverse property of the $\mathcal{F}$-component

$\mathcal{F}$ is the only component that relates to the original encryption-key bytes, and thus, it is critical that we analyze it. At each clock, the $\mathcal{F}$-component selects K_c and $K_{\tau_1(\alpha)}$ for further computation. K_c is simply selected by the counter c, while $K_{\tau_1(\alpha)}$ is selected based on the subscript $\tau_1(\alpha)$ which can be determined according to Eq. (2) and Table 1.

The inverse analysis of the $\mathcal{F}$-component aims at deducing the above two selected key bytes from the known output (O_0, O_1) and the feedback byte p. On rewriting the second formula of Eq. (1), K_c can be expressed by using O_1 and p as follows:

$$\begin{cases} K_{c,7} \oplus K_{c,3} = O_{1,3} \oplus p_7 \oplus p_3, \\ K_{c,6} \oplus K_{c,2} = O_{1,2} \oplus p_6 \oplus p_2, \\ K_{c,5} \oplus K_{c,1} = O_{1,1} \oplus p_5 \oplus p_1, \\ K_{c,4} \oplus K_{c,0} = O_{1,0} \oplus p_4 \oplus p_0. \end{cases} \tag{8}$$

For simplicity, let us use the following notations:

$$\begin{cases} \boldsymbol{k}_h = (K_{c,7}, K_{c,6}, K_{c,5}, K_{c,4})^{\mathrm{T}}, \\ \boldsymbol{k}_l = (K_{c,3}, K_{c,2}, K_{c,1}, K_{c,0})^{\mathrm{T}}, \\ \boldsymbol{p}_h = (p_7, p_6, p_5, p_4)^{\mathrm{T}}, \\ \boldsymbol{p}_l = (p_3, p_2, p_1, p_0)^{\mathrm{T}}. \end{cases}$$

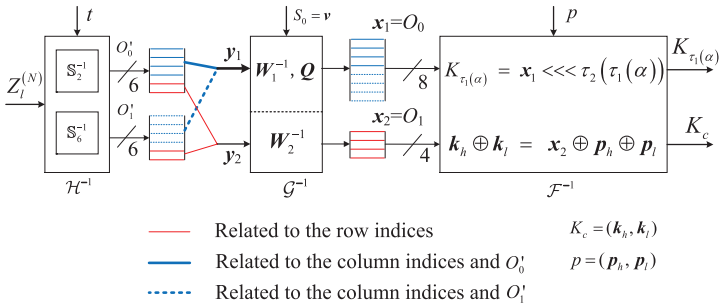

Figure 5 (Color online) Links among the three inverse components.

Thus, Eq. (8) becomes

$$k_h \oplus k_l = O_1 \oplus p_h \oplus p_l. \tag{9}$$

Therefore, for $i = 0, 1, 2, 3$, when $O_{1,i} \oplus p_{i+4} \oplus p_i = 0$, the candidate for $(K_{c,i+4}, K_{c,i})$ is selected from $\{(0,0),(1,1)\}$, and when $O_{1,i} \oplus p_{i+4} \oplus p_i = 1$, the candidate can only be selected from $\{(0,1),(1,0)\}$. This implies that given O_1 and p, Eq. (8) has 16 solutions for K_c.

Similarly, on rewriting the first formula of Eq. (1), $K_{\tau_1(\alpha)}$ can be obtained from O_0 using

$$K_{\tau_1(\alpha)} = O_0 \lll \tau_2(\tau_1(\alpha)), \tag{10}$$

where α is related to K_c and p and can be calculated on the basis of Eq. (2). This leads us to the following proposition.

Proposition 3. For the $\mathcal{F}$-component, if O_1 and p are known, then all possible values of K_c can be narrowed down from 2^8 to 2^4 according to Eq. (8). If O_0, p, and K_c are known, the input-key byte $K_{\tau_1(\alpha)}$ can be uniquely retrieved using Eq. (10).

Now we have obtained three inverse properties of the components of the GMR-2 cipher as described in Propositions 1–3. At the end of this subsection, we briefly discuss the links among these inverse components as depicted in Figure 5. Given the start point — a keystream byte $Z_l^{(N)}$ at the l-th clock with frame number N, let us assume that the feedback byte S_0 and p is known. On then using the $\mathcal{H}^{-1}$-component, 16 possible values of (O_0', O_1') are obtained. The $\mathcal{G}^{-1}$-component is subsequently used, which results in 16 different values of (O_0, O_1). Finally, after using the $\mathcal{F}^{-1}$-component, each possible value of (O_0, O_1) results in 16 candidates for $(K_c, K_{\tau_1(\alpha)})$. In total, one can obtain at most $16 \times 16 = 256$ possible values for $(K_c, K_{\tau_1(\alpha)})$. The details of the analysis are described in the following section.

4 Inverse properties of the GMR-2 cipher

In this section, we analyze how the three inverse components interact with each other and demonstrate the links between the keystream bytes and the original encryption-key bytes.

Given a frame number N, let $S_i^{(l)}$ denote the state of S_i at the l-th clock and $Z_l^{(N)}$ denote the keystream byte at the l-th clock with N-th frame in the keystream generation phrase, then for $8 \leqslant l \leqslant 14$ we have

$$S_0^{(l)} = Z_{l-8}^{(N)} \quad \text{and} \quad p = S_7^{(l)} = Z_{l-1}^{(N)},$$

which demonstrates that $S_0^{(l)}$ is equal to the keystream byte generated 8 clocks before, and p is equal to the last keystream byte. Hence, for $8 \leqslant l \leqslant 14$, both $S_0^{(l)}$ and p are known to us, and the vector $\boldsymbol{v}$ (as previously defined) is also known. We therefore only focus on the cipher at the $(c+8)$-th clock with $0 \leqslant c \leqslant 6$ in the following analysis. The main results obtained are the following two theorems.

Theorem 1. At the $(c+8)$-th clock with $0 \leqslant c \leqslant 6$, if K_c is known, then the corresponding encryption-key byte $K_{\tau_1(\alpha)}$ can be uniquely determined using the current keystream byte $Z_{c+8}^{(N)}$.

Proof. As p is known at the $(c+8)$-th clock, from Eq. (2), knowing K_c would aid in calculating α, as well as $\tau_1(\alpha)$ and $\tau_2(\tau_1(\alpha))$ by looking up Table 1. Moreover, O_1 (thus $\boldsymbol{x}_2$) can be obtained using Eq. (1), based on which $\boldsymbol{y}_2$ can be calculated from Eq. (6). Owing to Proposition 1, $\boldsymbol{y}_1$ which corresponds to the column indices for the two S-boxes can be uniquely determined from $Z_{c+8}^{(N)}$ and the row indices $\boldsymbol{y}_2$. Consequently, the value of O_0 can be uniquely determined using Proposition 2. Finally, with the help of Proposition 3, the value of $K_{\tau_1(\alpha)}$ can be calculated definitely from O_0, K_c and p.

Theorem 2. At the $(c+8)$-th clock with $0 \leqslant c \leqslant 6$, each keystream byte $Z_{c+8}^{(N)}$ exactly corresponds to 256 possible values of the triple $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$, where K_c ranges from 0 to 255.

Proof. Firstly, according to Propositions 1 and 2, each keystream byte $Z_{c+8}^{(N)}$ corresponds to 16 different O_1, and for each O_1, Proposition 3 further indicates the existence of 16 different candidates for K_c.

Next, by contradiction, we can prove that the candidates for K_c obtained by different O_1 will be different from each other. That is to say, assuming that $O_1^{(i)} \neq O_1^{(j)}$ holds, one can state that the candidates $K_c^{(i)}$ and $K_c^{(j)}$ that are derived from $O_1^{(i)}$ and $O_1^{(j)}$ must be different. Otherwise, if $K_c^{(i)} = K_c^{(j)}$, then Eq. (1) indicates that $O_1^{(i)} = O_1^{(j)}$, which contradicts the aforementioned hypothesis.

The above two steps demonstrate that each keystream byte $Z_{c+8}^{(N)}$ exactly corresponds to 256 values of K_c. Moreover, through Theorem 1, $\tau_1(\alpha)$ and $K_{\tau_1(\alpha)}$ can be uniquely obtained from the K_c and $Z_{c+8}^{(N)}$, which completes this proof.

Theorem 2 shows that each $Z_{c+8}^{(N)}$ can be used to derive several triples $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$ for $0 \leqslant c \leqslant 7$. Next, how these triples can be further used to obtain new information on K is discussed. Here we list two rules that are crucial to our attack.

Rule 1. Given one triple $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$ corresponding to the keystream byte $Z_{c+8}^{(N)}$ with $0 \leqslant c \leqslant 6$, if $\tau_1(\alpha) = c$, we can compare $K_{\tau_1(\alpha)}$ with K_c:

(i) If $K_c = K_{\tau_1(\alpha)}$, it indicates that such a value of K_c can be regarded as a candidate.

(ii) If $K_c \neq K_{\tau_1(\alpha)}$, it means that such a value of K_c cannot be a candidate and should be discarded.

Rule 2. Given two triples $\left(K_m, K_{\tau_1(\alpha_m)}, \tau_1(\alpha_m)\right)$ and $\left(K_n, K_{\tau_1(\alpha_n)}, \tau_1(\alpha_n)\right)$ which correspond to $Z_{m+8}^{(N)}$ and $Z_{n+8}^{(N)}$ with $0 \leqslant m \neq n \leqslant 8$:

(i) If $\tau_1(\alpha_n) = m$, we can compare $K_{\tau_1(\alpha_n)}$ and K_m:

- If $K_{\tau_1(\alpha_n)} = K_m$, it indicates that such a value of (K_m, K_n) can be regarded as a candidate.
- If $K_{\tau_1(\alpha_n)} \neq K_m$, such a value of (K_m, K_n) cannot be a candidate and should be discarded.

(ii) If $\tau_1(\alpha_n) = \tau_1(\alpha_m)$, we can compare $K_{\tau_1(\alpha_n)}$ and $K_{\tau_1(\alpha_m)}$:

- If $K_{\tau_1(\alpha_n)} = K_{\tau_1(\alpha_m)}$, it indicates that such a value of (K_m, K_n) can be regarded as a candidate.
- If $K_{\tau_1(\alpha_n)} \neq K_{\tau_1(\alpha_m)}$, such a value of (K_m, K_n) cannot be a candidate and should be discarded.

5 The real-time inversion attack on the GMR-2 cipher

In this section, we present a very efficient and practical attack against the GMR-2 cipher with low time and data complexity. We call this attack the real-time inversion attack.

5.1 An overview of the inversion attack

As shown in Figure 6, we first briefly explain the inversion attack procedure, which is divided into the following three phases.

Phase 1: table generation. Intercept a certain number of keystream bytes (usually only one frame is sufficient) and then adopt Theorem 2 to generate the seven lists which map the keystream bytes at the $(c+8)$-th clock with $0 \leqslant c \leqslant 6$ to the original key bytes. Meanwhile, build a virtual list for the seventh original-key byte K_7. We refer to these lists as tables.

Phase 2: dynamic table look-up, filtration and combination. Alternately perform table look-up (8 lists) and the filtration procedure (through Rules 1 and 2) to construct valid key chains (the definition is provided in Subsection 5.3; it is a special form of candidate key byte), and combine them to form the full 8-byte candidate keys.

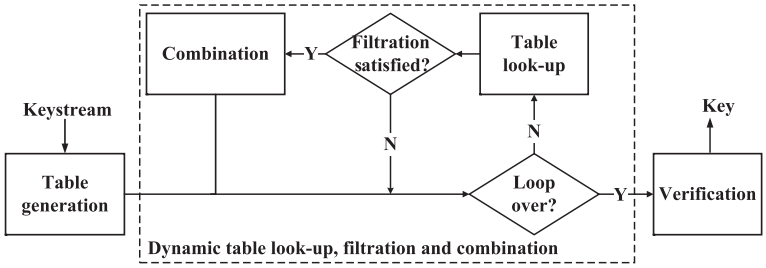

Figure 6 Overview of the inversion attack procedure.

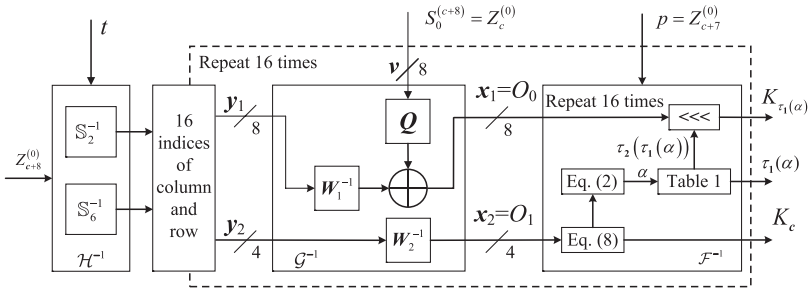

Figure 7 Table generation procedure.

Phase 3: verification. Verify the correctness of those candidate keys obtained in Phase 2 using the intercepted keystream bytes (usually the first 8 bytes of a frame are sufficient), discard all wrong 8-byte encryption-keys.

5.2 Phase 1: table generation

Without the loss of generality, let us assume the frame number of the keystream bytes is $N = 0$, and let $(Z_0^{(0)}, Z_1^{(0)}, \ldots, Z_{14}^{(0)})$ denote the known 15 bytes of keystream. In order to ensure that the values of p and $S_0 = \boldsymbol{v}$ are known, we analyze the cipher at the $(c+8)$-th clock for $0 \leqslant c \leqslant 6$.

According to the mechanism of the GMR-2 cipher, each keystream byte $Z_{c+8}^{(0)}$ is related with

$$\Big(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha), p, S_0^{(c+8)}, t\Big) = \Big(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha), Z_{c+7}^{(0)}, Z_c^{(0)}, c \bmod 2\Big),$$

which means that a mapping between $(Z_{c+8}^{(0)}, Z_{c+7}^{(0)}, Z_c^{(0)})$ and $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$ can be established if c is known. Thus, from the known keystream $(Z_0^{(0)}, Z_1^{(0)}, \ldots, Z_{14}^{(0)})$, we can obtain seven groups of $(Z_{c+8}^{(0)}, Z_{c+7}^{(0)}, Z_c^{(0)})$ for $0 \leqslant c \leqslant 6$, and each group can be used to build 256 possible values of the triple $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$ based on Theorem 2.

For a better explanation, one can refer the table generation procedure in Figure 7. During this phase, for each group $(Z_{c+8}^{(0)}, Z_{c+7}^{(0)}, Z_c^{(0)})$ with $0 \leqslant c \leqslant 6$, the following steps are performed:

(i) Look up the two S-boxes in order to obtain the 16 values of $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ using the keystream byte $Z_{c+8}^{(0)}$ and the toggle-bit t.

(ii) Calculate the corresponding values of $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ using Eq. (7) for a given $(\boldsymbol{y}_1, \boldsymbol{y}_2)$ from step (i), this also corresponds to O_0 and O_1.

(iii) Find 16 different values of K_c for a given O_1 according to Eq. (8), and then obtain the related values of $K_{\tau_1(\alpha)}$ and $\tau_1(\alpha)$ according to Theorem 1, which yields 16 triples $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$.

(iv) Repeat Steps (ii) and (iii) for 16 different values of $(\boldsymbol{y}_1, \boldsymbol{y}_2)$, thereby yielding 256 triples $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$ that are stored in a list denoted by $\mathcal{L}_c$ in which K_c is sorted in ascending order.

It should be noted that the above table-generation procedure cannot deduce information for the seventh original-key byte K_7 from the known keystream $(Z_0^{(0)}, Z_1^{(0)}, \ldots, Z_{14}^{(0)})$, i.e., we can only assume that the candidates for K_7 range from 0 to 255, but the corresponding values of $K_{\tau_1(\alpha)}$ and $\tau_1(\alpha)$ are not available.

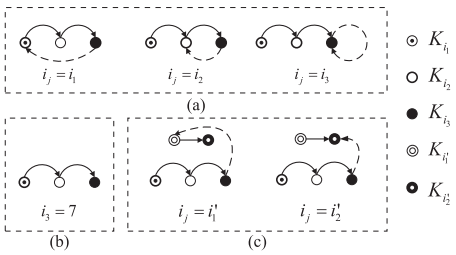

Figure 8 Diagram of the links for valid key chains in Example 1. (a) Case (1) in Definition 2; (b) case (2) in Definition 2; (c) case (3) in Definition 2.

Thus we build a virtual list for K_7 ranging from 0 to 255 but with empty values for $K_{\tau_1(\alpha)}$ and $\tau_1(\alpha)$. In total, we generate eight lists, each containing 256 triples. These eight lists are denoted by

$$\{\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4, \mathcal{L}_5, \mathcal{L}_6, \mathcal{L}_7\}.$$

5.3 Phase 2: dynamic table look-up, filtration and combination

Now we have generated eight lists in Phase 1; however, simply attempting an exhaustive search using these lists without any strategy offers no advantage over the brute-force attack. Thus, before describing our proposed inversion attack strategy, we first introduce the concept of the "valid key chain" based on the eight lists generated in Phase 1. The core idea of this concept is that one can use it to link the candidate-key bytes obtained separately from the eight lists in a chain style such that Rules 1 and 2, which are presented in Section 4, can be adopted to filter out wrong candidates efficiently.

Definition 1 (Key chain)**.** A sequence of ordered key bytes:

$$(\,(i_1, K_{i_1}), (i_2, K_{i_2}), \ldots, (i_l, K_{i_l})\,)$$

with different i_j $(1 \leqslant j \leqslant l)$, where i_j is the index (subscript) for K_{i_j}, is called a key chain with a length of l bytes if it satisfies the following condition: for every $1 \leqslant m \leqslant l-1$, there exists a list $\mathcal{L}_{i_m}$ such that $(K_{i_m}, K_{i_{m+1}}, i_{m+1}) \in \mathcal{L}_{i_m}$. For convenience, we simply use

$$K_{i_1} \to K_{i_2} \to \cdots \to K_{i_l}$$

to denote this key chain, where K_{i_1} is the starting node and K_{i_l} is the ending node.

Definition 2 (Valid key chain)**.** A key chain $K_{i_1} \to K_{i_2} \to \cdots \to K_{i_l}$ with a length of l bytes is called a valid key chain if it satisfies one of the following conditions:

(1) There exists an index $i_j \in \{i_1, i_2, \ldots, i_l\}$ such that $(K_{i_l}, K_{i_j}, i_j) \in \mathcal{L}_{i_l}$.

(2) $i_l = 7$ and there is no other valid key chain that contains the key byte K_7.

(3) There already exists a valid key chain with a length of n bytes: $K_{i'_1} \to K_{i'_2} \to \cdots \to K_{i'_n}$; furthermore, there exists an index $i_j \in \{i'_1, i'_2, \ldots, i'_n\}$ such that $(K_{i_l}, K_{i_j}, i_j) \in \mathcal{L}_{i_l}$.

Definition 3 (Disjoint valid key chains)**.** Two valid key chains $\mathcal{C}_1$ and $\mathcal{C}_2$ are said to be disjoint, if they satisfy the condition that for each node $K_{i_m} \in \mathcal{C}_1$ and each node $K_{j_n} \in \mathcal{C}_2$, their subscripts are different, i.e., $i_m \neq j_n$.

As the definition of valid key chain is dependent on the eight lists generated in Phase 1, to further understand this concept as well as its properties, we provide the following example as an illustration.

Example 1. Given a key chain with a length of three bytes (all the subscripts for the key bytes are different): $K_{i_1} \to K_{i_2} \to K_{i_3}$; the following three cases imply three types of valid key chains:

(1) There exists an index $i_j \in \{i_1, i_2, i_3\}$ such that $(K_{i_3}, K_{i_j}, i_j) \in \mathcal{L}_{i_3}$, as show in Figure 8(a).

(2) $i_3 = 7$, and there is no other valid key chain that comprises K_7, as show in Figure 8(b).

(3) There already exists a valid key chain with a length of 2 bytes: $K_{i'_1} \to K_{i'_2}$; furthermore, there exists an index $i_j \in \{i'_1, i'_2\}$ such that $(K_{i_3}, K_{i_j}, i_j) \in \mathcal{L}_{i_3}$, as show in Figure 8(c).

Table 4 Definitions of the variables and candidate sets

Variable	Definition	Initialization
$\mathcal{R}$	The $(i-1)$ valid key chains obtained before: $\mathcal{R}=\{\gamma_1,\gamma_2,\ldots,\gamma_{i-1}\}$.	$\emptyset$
Δ	The key chain currently being looked up: $\Delta=\gamma_i=(\delta_{i_1}^{(i)}\to\delta_{i_2}^{(i)}\to\cdots\to\delta_{i_{l-1}}^{(i)})$.	$\emptyset$
$\Gamma=\{\Gamma_1,\Gamma_2\}$	The set of indices (subscripts) for the key bytes that has been obtained by table look-up, where Γ_1 corresponds to the key bytes in $\mathcal{R}$, and Γ_2 for key bytes in Δ.	$\emptyset$
KC	The candidate set of the complete 8-byte encryption keys.	$\emptyset$
(c,K_c)	Query point that queries the K_c-th row in the c-th list $\mathcal{L}_c$, it is also used as the control parameter for ending Phase 2.	$(0,0)$

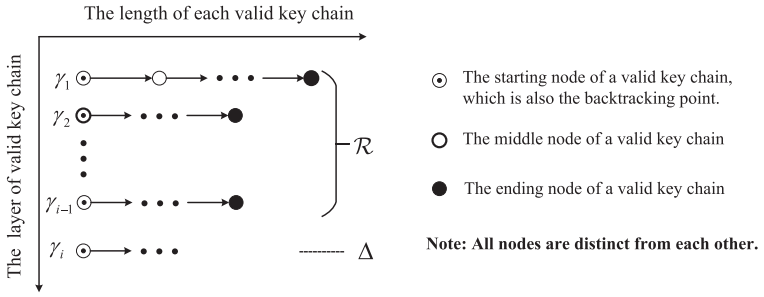

Figure 9 Dynamic combination of valid key chains in Phase 2 (the number of valid key chain layers as well as the length of each valid key chain are dynamically changed).

Properties of valid key chains. According to the definition, for the GMR-2 cipher, the minimum length of a valid key chain is 1 byte, which means that the key byte is associated with itself, and this situation corresponds to the "read-collision" case in [9]; while the maximum length of a valid key chain is 8 bytes, which means that all the 8 key bytes are connected in one chain. Therefore, an 8-byte encryption key can be divide into at most 8 valid key chains, each containing just 1 key byte, or at least 1 valid key chain containing the entire 8 key bytes. In general, one full 8-byte encryption key can be decomposed into i disjoint valid key chains with $1\leqslant i\leqslant 8$ (this decomposition is finally combined to form an entire 8-byte candidate key in the attack procedure of Phase 2).

Main idea of Phase 2. Using the concept of the valid key chain, Phase 2 can be described as "dynamically seeking all valid key chains (that accord with Rules 1 and 2 on looking up table and performing filtration) and combining them to form candidates for the complete 8-byte encryption key". Let us define three candidate sets $\mathcal{R}$, Δ and KC, an index set Γ and a query point (c,K_c) as shown in Table 4. Using these symbols, and referring to Figure 9, the second phase of the inversion attack can be briefly explained as follows (for the details of the attack procedure of Phase 2, one can refer to the algorithms described in Appendixes A and B):

(i) Choose a starting node (query point) $(0,K_0)$, and for each possible value of K_0 (in the range of 0–255), dynamically look up the table (8 lists obtained in Phase 1) in a serialized manner in order to build up a key chain Δ, and store the indices (subscripts) for the key bytes obtained in Δ into the set Γ_2. Once Δ becomes "valid" through the filtration procedure, treat $\Delta=\gamma_1$ as the first layer of the valid key chain, and store the key bytes of the chain as well as their indices (subscripts) into $\mathcal{R}$, and copy these indices (subscripts) into Γ_1.

(ii) Choose a new starting node $(\min(\overline{\Gamma}),K_{\min(\overline{\Gamma})})$ where $\min(\overline{\Gamma})$ is the minimum subscript for the key bytes (K_0–K_7) that have not been previously obtained[3)] (by looking up the relevant table), and for each possible value of such a key byte (in the range of 0–255), continue to lookup the table and perform the

3) As each time the starting node is chosen in such a way and, according to Definitions 2, these constructed valid key chains, which are finally combined to form a full 8-byte candidate encryption key, are disjoint.

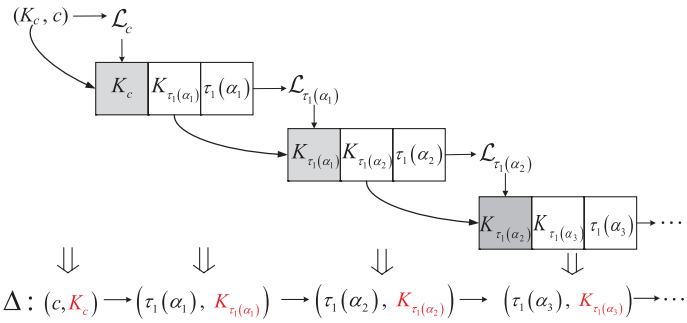

Figure 10 (Color online) Procedure of dynamic table look-up in Phase 2.

filtration alternately to build up the i-th layer of a valid key chain $\Delta = \gamma_i$ for $1 \leqslant i \leqslant 8$. Similarly, update the sets $\mathcal{R} = \{\gamma_1, \gamma_2, \ldots, \gamma_i\}$, Γ_1 and Γ_2.

(iii) Check whether all the valid key chains in $\mathcal{R}$ exactly cover the entire 8-byte encryption key, and if so, combine these valid key chains, save them in KC, and backtrack to the starting node of Δ to find a new valid key chain (in order to find new candidate keys). Else return to Step (ii).

(iv) Repeat Steps (i)–(iii) until all the candidate 8-byte encryption keys are obtained.

Procedure for dynamically looking up table to build up a key chain. Given a query point (c, K_c) as the starting node of a chain Δ (refer to Figure 10), we can use c to point to the list $\mathcal{L}_c$, which is then used by the adversary to look up in order to obtain $(K_{\tau_1(\alpha_1)}, \tau_1(\alpha_1))$ corresponding to the row value K_c. This is followed by a second similar procedure; at this point, we have obtained a middle node $(\tau_1(\alpha_1), K_{\tau_1(\alpha_1)})$, then we can use $\tau_1(\alpha_1)$ to point to the list $\mathcal{L}_{\tau_1(\alpha_1)}$, which indicates a new result $(K_{\tau_1(\alpha_2)}, \tau_1(\alpha_2))$ by looking-up its row value $K_{\tau_1(\alpha_1)}$. On repeating this process, we can further obtain the next middle nodes $(K_{\tau_1(\alpha_3)}, \tau_1(\alpha_3)) \cdots$ by using the list $\mathcal{L}_{\tau_1(\alpha_2)} \cdots$, and we thus obtain a key chain:

$$\Delta = (c, K_c) \to \big(\tau_1(\alpha_1), K_{\tau_1(\alpha_1)}\big) \to \big(\tau_1(\alpha_2), K_{\tau_1(\alpha_1)}\big) \to \big(\tau_1(\alpha_3), K_{\tau_1(\alpha_3)}\big) \to \cdots,$$

which is then passed to the filtration procedure to check whether it is a valid key chain.

Procedure for performing filtration to obtain a valid key chain. The purpose of the filtration is to check when the key chain obtained by looking up the table will be a valid key; this can be achieved by applying Rules 1 and 2 in order to discard the inconsistent cases. Moreover, during the filtration, we are required to perform the following backtracking steps as well:

- If the ending node of a key chain disagrees with the constraints of Rules 1 and 2, such a chain would not form a valid key chain. We then backtrack to the starting node of the current key chain $\Delta = \gamma_i$, update $\Gamma_2 \leftarrow \emptyset$, $\Delta \leftarrow \emptyset$, set a new value for this starting node (as the query point), and then perform a similar procedure of dynamically looking up the relevant table, filtration and combination.
- If the starting node of $\Delta = \gamma_i$ exceeds the range of 0–255, we backtrack to the starting node of the $(i-1)$-th layer of the valid key chain γ_{i-1} in $\mathcal{R}$ and perform a similar procedure. This procedure is repeated until we backtrack to the first layer of the valid key chain γ_1. If the starting node of γ_1 is exceeds the range of 0–255, which indicates that all the valid key chains have been found, we then stop Phase 2 of the inversion attack.

5.4 Phase 3: verification

In order to exclude wrong candidate keys, Phase 3 tests the candidate keys stored in KC one by one using the first 8 bytes $(Z_0^{(0)}, Z_1^{(0)}, \ldots, Z_7^{(0)})$ of the known keystream. For each candidate key, the following steps are performed:

(i) Fill the key register K with the candidate key, and initialize the shift register S with the known frame number.

(ii) Clock the cipher 8 times for initialization, and obtain the next 8 bytes keystream.

(iii) Compare this calculated keystream with the corresponding 8-byte of the intercepted known keystream. If they match, the correct key is obtained; otherwise, this candidate key is discarded.

6 Experimental results and complexity analysis

In order to verify the validity of our proposed attack, in this section, we present some experiments and a complexity analysis.

6.1 Experimental results

We carried out 10000 experiments using a 3.3-GHz PC for the GMR-2 cipher with random frame numbers and keys. Our results demonstrate that the retrieved encryption key may not be unique for a known 15-byte keystream at some cases. In other words, there exist multiple encryption keys corresponding to the same 15-byte keystream, and these encryption keys usually differs only one byte from each other. More precisely, each 15-byte keystream indicates 1.03 encryption keys on average, of which approximately 97.2% of the keystreams indicate a unique encryption key, and the remaining 2.8% keystreams indicate multiple (at most four) encryption keys. Thus, in order to overcome this problem, one additional keystream byte of another frame is required in these cases, which means that 9 bytes of the keystream are completely exploited in the third phase. Therefore, an additional frame of the keystream is leveraged in Phase 1, and the required number of keystream bytes for entire attack is 15–16.

For the comparison, the frequency distribution of the number of the candidate keys in Phase 2 for each attack is plotted with respect to an average number of 7755, and is shown in Figure 11. It can be observed from this figure that a verification is required to be performed 7755 times on average during Phase 3. Meanwhile, the time consumed for each attack is also obtained with the distribution shown in Figure 12. It can be observed that the 8-byte encryption key can be derived in approximately 0.02 s on average, where 0.08 ms is consumed for generating the table, 3.37 ms is consumed for verifying the candidates, and the remaining 16.55 ms is consumed in Phase 2.

We also point out that if we perform the forward verification each time an 8-byte candidate key is combined during Phase 2 which involves alternating Phases 2 and 3 at the same time; then once an 8-byte candidate key passes the forward verification of the 9 bytes of the keystream, the attack can be stopped. In this case, we can accelerate the inversion attack. In this optimized inversion attack, the average number of verifications to be performed is reduced to 3980 and the time consumed is approximately 0.01 s on average. This optimized attack procedure is presented in Figure 13.

6.2 Complexity analysis

Time complexity analysis. The time complexity of our inversion attack takes into consideration the time required for table generation, dynamic table look-up, filtration, combination, as well as the verification. The time complexity can be analysed from the experimental statistics (see Subsection 6.1). However, for convenience, we only focus on the exhaustive search space. As we perform the verification $7755 \approx 2^{13}$ times on average, the size of the exhaustive search space is thus approximately 2^{13}, which could be further reduced to $3980 \approx 2^{12}$ on average by adopting the optimized attack.

Data complexity analysis. The data complexity of our attack is 15–16 bytes of the keystream. In 10000 experiments, approximately 97.2% of the encryption keys can be uniquely determined using the 15 bytes of the keystream, and the rest (approximately 2.8%) of the cases require an extra keystream byte. Thus, $15 \times 97.2\% + 16 \times 2.8\% \approx 15.03$ bytes of the keystream are required to distinguish the correct encryption key from 2^{13} candidates on average.

Memory complexity analysis. The memory complexity of our attack stems mainly from the table (eight lists) generated in Phase 1. As each list comprises 256 triples $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$, our attack requires approximately $256 \times 3 \times 8$ bytes = 6 KB of storage space.

7 Conclusion

In this paper, we propose a very efficient, real-time inversion attack against the GMR-2 cipher. It can retrieve the complete 8-byte encryption key using only one frame (15 bytes) of the keystream on average,

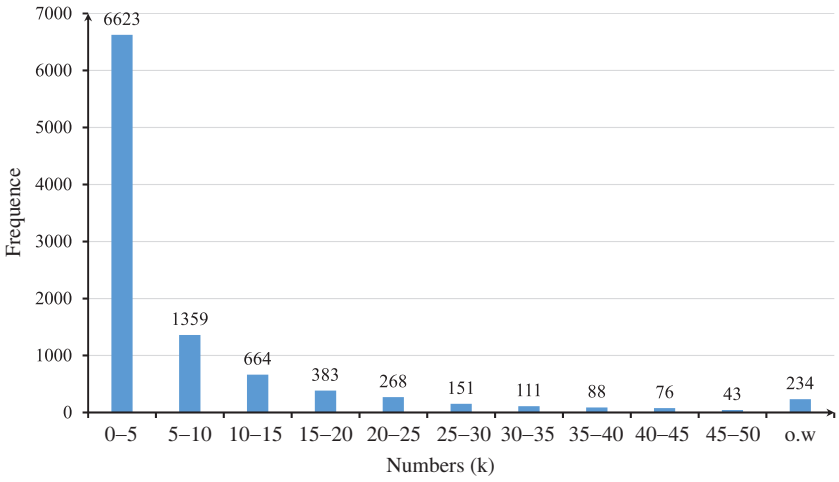

Figure 11 (Color online) Frequency distribution of the number of candidate keys in Phase 2 (the numbers on the horizontal axis are in thousand of times, and each interval a–b contains the left number a, but does not contain the right value b).

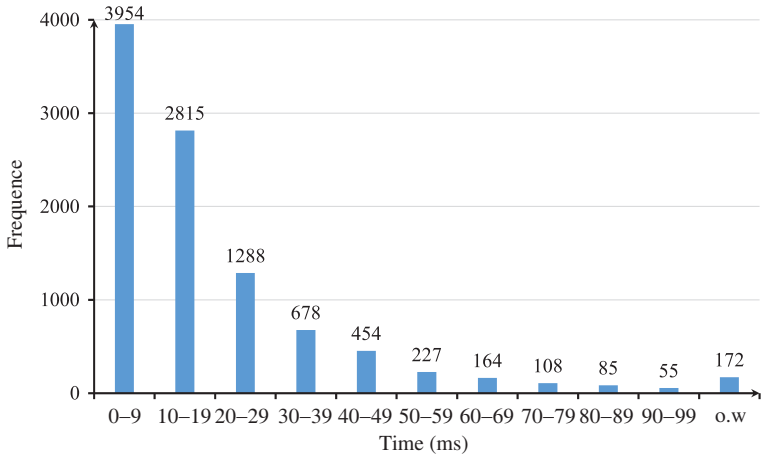

Figure 12 (Color online) Frequency distribution of the attack time.

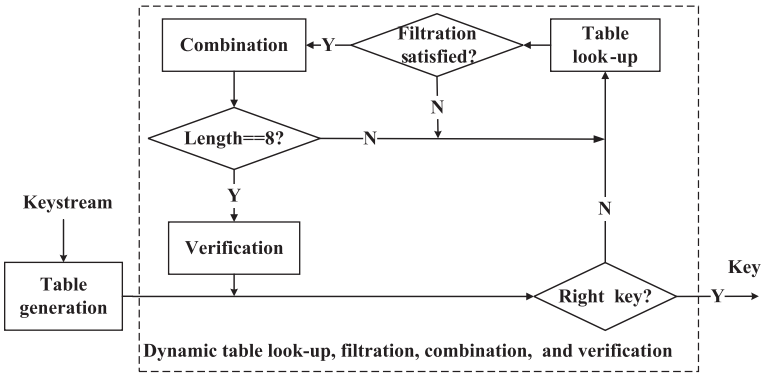

Figure 13 Optimized inversion attack procedure.

the size of the exhaustive search space can be reduced to approximately 2^{13}, and the memory complexity is 6 KB.

Table 5 shows the comparison between the known cryptanalytic results and our results, from which it can be observed that the inversion attack proposed in this paper possesses evident superiority as compared to the dynamic guess-and-determine and the read-collision based attacks. Given one frame (15 bytes) of the keystream, one can break the GMR-2 cipher within only 0.02 s using a 3.3-GHz PC. This further

Table 5 Cryptanalytic results for the GMR-2 cipher

Method	Data	Brute force space	Memory	Average time
Read-collision based technique [9]	15–20 frames	2^{10}	$\sim$	-
Read-collision based technique [9]	4–5 frames	2^{18}	$\sim$	-
Dynamic guess-and-determine [13]	1 frame	2^{28}	$\sim$	280 s$^{\blacktriangle}$
Inversion attack (this paper)	1 frame	2^{13}	6 KB	0.02 s$^{\triangle}$
Optimized inversion attack (this paper)	1 frame	2^{12}	6 KB	0.01 s$^{\triangle}$

▲: Experimental plantform: 3.3-GHz PC; Number of experiments: 1000.

△: Experimental plantform: 3.3-GHz PC; Number of experiments: 10000.

demonstrates that there exists serious security flaws in the GMR-2 cipher, and it is crucial for service providers to upgrade the cryptographic modules[4)] of the satellite phone system in order to provide the communication confidentiality.

Acknowledgements The authors wish to thank the anonymous reviewers for their valuable suggestions and comments, which greatly improve the presentation and quality of the current paper. This work in this paper was supported by National Nature Science Foundation of China (Grant Nos. 61402515, 61672530).

References

1 ETSI TS. GEO-Mobile Radio Interface Specifications. 2001
2 Biryukov A, Shamir A, Wagner D. Real time cryptanalysis of A5/1 on a PC. In: Proceedings of the 7th International Workshop on Fast Software Encryption. Berlin: Springer, 2000. 1–18
3 Dunkelman O, Keller N, Shamir A. A practical-time attack on the A5/3 cryptosystem used in third generation GSM telephony. In: Proceedings of Annual Cryptology Conference, Santa Barbara, 2010. 393–410
4 Kircanski A, Youssef A M. On the sliding property of SNOW 3G and SNOW 2.0. IET Inf Secur, 2011, 5: 199–206
5 Li L, Liu X H, Wang Z, et al. An improved attack on clock-controlled shift registers based on hardware implementation. Sci China Inf Sci, 2013, 56: 112107
6 Wu H J, Huang T, Nguyen P H, et al. Differential attacks against stream cipher ZUC. In: Proceedings of the 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, 2012. 262–277
7 Zhang B, Xu C, Meier W. Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of SNOW 2.0. In: Proceedings of Annual Cryptology Conference, Santa Barbara, 2015. 643–662
8 Zhou C F, Feng X T, Lin D D. The initialization stage analysis of ZUC v1.5. In: Proceedings of International Conference on Cryptology and Network Security, Sanya, 2011. 40–53
9 Driessen B, Hund R, Willems C, et al. Don't trust satellite phones: a security analysis of two satphone standards. In: Proceedings of IEEE Symposium on Security and Privacy (SP), Oakland, 2012. 128–142
10 Driessen B, Hund R, Willems C, et al. An experimental security analysis of two satphone standards. ACM Trans Inf Syst Secur, 2013, 16: 10
11 Barkan P, Biham E, Keller N. Instant cipher-text only cryptanalysis of GSM encrypted communication. J Cryptol, 2008, 21: 392–429
12 Bogdanov A, Eisenbarth T, Rupp A. A hardware assisted real-time attack on A5/2 without precomputations. In: Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems, Vienna, 2007. 394–412
13 Li R L, Li H, Li C, et al. A low data complexity attack on the GMR-2 cipher used in the satellite phones. In: Proceedings of International Workshop on Fast Software Encryption, Singapore, 2013. 485–501
14 Golic J D. On the security of nonlinear filter generators. In: Proceedings of the 3rd International Workshop on Fast Software Encryption, Cambridge, 1996. 173–188
15 Golic J D, Clark A, Dawson E. Inversion attack and branching. In: Proceedings of Australasian Conference on Information Security and Privacy, Wollongong, 1999. 99–102
16 Golic J D, Clark A, Dawson E. Generalized inversion attack on nonlinear filter generators. IEEE Trans Comput, 2000, 49: 1100–1109

4) It should be noted that the GMR-2 cipher is currently being used in the "IsatPhone Pro" satellite phones.

Appendix A Phase 2 (Part I) of the inversion attack

Algorithm A1 Inversion Attack: Phase 2 (Part I)

Input: Keystream-related lists $\{\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4, \mathcal{L}_5, \mathcal{L}_6, \mathcal{L}_7\}$.
Output: Key candidate set **KC**.
Initialization: $\mathcal{R} \leftarrow \emptyset$;$\Delta \leftarrow \emptyset$;$\Gamma \leftarrow \emptyset$;$(c, K_c) \leftarrow (0, 0)$; KC $\leftarrow \emptyset$.
repeat
  $\left(\tau_1(\alpha), K_{\tau_1(\alpha)}\right) \leftarrow \text{LookUpTable}(c, K_c, \mathcal{L}_c)$;
  if $\tau_1(\alpha) \in \Gamma$ **then** /* Given that the $\tau_1(\alpha)$-th key byte $K_{\tau_1(\alpha)}$ already exists in the candidate sets, perform the filtration using Rule 2 */
    if $\left(\tau_1(\alpha), K_{\tau_1(\alpha)}\right) \in \Delta$ **or** $\left(\tau_1(\alpha), K_{\tau_1(\alpha)}\right)$ belongs to a certain valid key chain of $\mathcal{R}$ **then**
      $\Gamma_2 \leftarrow \Gamma_2 \cup \{c\}$; $\Delta \leftarrow \Delta \cup \{(c, K_c)\}$;
      $(c, K_c) \leftarrow \text{Combine}(\Delta)$; /* The valid key chain obtained at this time agrees with case (1) or (3) in Definition 2. */
    else
      $(c, K_c) \leftarrow \text{BackTrack}(\Delta)$;
    end
  else
    if $\tau_1(\alpha) == c$ **then**
      if $K_{\tau_1}(\alpha) == K_c$ **then** /* Perform the filtration using Rule 1. */
        $\Gamma_2 \leftarrow \Gamma_2 \cup \{c\}$; $\Delta \leftarrow \Delta \cup \{(c, K_c)\}$;
        $(c, K_c) \leftarrow \text{Combine}(\Delta)$; /* The valid key chain obtained at this time agrees with case (1) in Definition 2. */
      else
        $(c, K_c) \leftarrow \text{BackTrack}(\Delta)$;
      end
    else
      if $\tau_1(\alpha) == 7$ **then**
        $\Gamma_2 \leftarrow \Gamma_2 \cup \{c, 7\}$; $\Delta \leftarrow \Delta \cup \{(c, K_c), (7, K_7)\}$;
        $(c, K_c) \leftarrow \text{Combine}(\Delta)$; /* The valid key chain obtained at this time agrees with case (2) in Definition 2. */
      else /* Continue to find the next node of the current key chain. */
        $\Delta \leftarrow \Delta \cup \{(c, K_c)\}$;
        $(c, K_c) \leftarrow (\tau_1(\alpha), K_{\tau_1}(\alpha))$;
      end
    end
  end
until $K_c > 255$ **and** $c = 0$;
return KC;

Appendix B Phase 2 (Part II) of the inversion attack

Algorithm B1 Inversion Attack: Phase 2 (Part II)

Function `Combine(`Δ`)` `/* Combine a valid key chain or the entire 8-byte key, and return the new starting point (query point) of a new key chain. */`

$\mathcal{R} \leftarrow \mathcal{R} \cup \{\Delta\}$; $\Gamma_1 \leftarrow \Gamma_1 \cup \Gamma_2$;
if Length$(\Gamma_1) == 8$ **then** `/* In this case, we have obtained a complete 8-byte candidate key, and thus, we save it in KC and backtrack to the starting node in order to find another candidate key. */`
KC $\leftarrow$ KC $\cup \mathcal{R}$;
$\mathcal{R} \leftarrow \mathcal{R} - \{\Delta\}$; $\Gamma_1 \leftarrow \Gamma_1 - \Gamma_2$;
$(c, K_c) \leftarrow$ BackTrack(Δ);
else `/* If not, we seek the next valid key chain. Here,` $\overline{\Gamma}$ `denotes the set of indices (subscripts) for the key bytes that have not been previously obtained. */`
$(c, K_c) \leftarrow \left(\min\left(\overline{\Gamma}\right), 0\right)$;
end
$\Gamma_2 \leftarrow \emptyset$; $\Delta \leftarrow \emptyset$;
return (c, K_c);
end

Function `BackTrack(`Δ`)`

$(c, K_c) \leftarrow$ StartingNodeOf(Δ); `/* Backtrack to the starting node of the current key chain` γ_i`. */`
$K_c \leftarrow K_c + 1$; `/* Update the key value of the starting node of` γ_i`. */`
if $K_c > 255$ **then**
if $c = 0$ **then**
return (c, K_c);
end
$\Delta \leftarrow \gamma_{i-1}$; $\Gamma_2 \leftarrow$ SubscriptOf(Δ); `/* Update the current key chain. */`
$\mathcal{R} \leftarrow \mathcal{R} - \{\Delta\}$; $\Gamma_1 \leftarrow \Gamma_1 - \Gamma_2$;
$(c, K_c) \leftarrow$ BackTrack(Δ); `/* Backtrack to the starting node of the previous valid key chain` γ_{i-1}`. */`
end
$\Gamma_2 \leftarrow \emptyset$; $\Delta \leftarrow \emptyset$;
return (c, K_c);
end