IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 March 2023, accepted 24 April 2023, date of publication 1 May 2023, date of current version 10 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3271994

RESEARCH ARTICLE

A Practical Ciphertext-Only Attack on GMR-2 System

DONGJAE LEE[1], JAEWOO KIM[2], DEUKJO HONG[3], JAECHUL SUNG[4], AND SEOKHIE HONG[1]

[1]Institute of Cyber Security and Privacy (ICSP), Korea University, Seoul 02841, Republic of Korea
[2]Department of Computer Science and Engineering, Seoul National University, Seoul 08826, Republic of Korea
[3]Department of Information Technology and Engineering, Jeonbuk National University, Jeonju 54896, Republic of Korea
[4]Department of Mathematics, University of Seoul, Seoul 02504, Republic of Korea

Corresponding author: Seokhie Hong (shhong@korea.ac.kr)

This work was supported in part by the Military Crypto Research Center through the Defense Acquisition Program Administration (DAPA) and the Agency for Defense Development (ADD) under Grant UD210027XD.

ABSTRACT We present a ciphertext-only attack on the GEO-Mobile Radio Interface-2 (GMR-2) system for the first time. The GMR-2 is a satellite communication standard adopted by Inmarsat, a British satellite telecommunications company that offers global mobile services. The best publicly known attack on GMR-2 is a known plaintext attack called the inversion attack, proposed by Hu et al. in 2018. It recovers the 64-bit session key in 20 milliseconds when one keystream frame (15-byte) is available. Our contributions are twofold. First, we improve the previous inversion attack using a novel approach, pre-filtration. With our improvement, we can recover the session key in 4.5 milliseconds and 0.62 milliseconds using one and two keystream frames, respectively. Second, we propose a practical ciphertext-only attack on the GMR-2 by exploiting a vulnerability in the CIPHERING MODE COMMAND message type. We find that this message type only has 2^{11} degrees of freedom despite being transmitted in a 184-bit format. Additionally, we find that two or more keystream frames can be derived from a single message in four of the six channels through which this message type may be transmitted. Assuming the CIPHERING MODE COMMAND message type is transmitted using one of these four channels, we can iteratively guess the message and conduct a known plaintext attack to recover the session key. Thanks to the speed improvement achieved by our pre-filtration method, our ciphertext-only attack can recover the session key in 1.3 seconds.

INDEX TERMS A5-GMR-2 stream cipher, ciphertext-only attack, cryptography, Inmarsat, GMR-2 satellite communication system.

I. INTRODUCTION

People in areas where terrestrial communication systems are unavailable can communicate using satellite communication systems, which can offer mobile services regardless of regional limitations. They can cover sparsely populated areas and harsh environments, such as the ocean, desert, and mountainous areas. Satellite communications typically use the GEO-Mobile Radio Interface (GMR) standard from the European Telecommunication Standards Institute (ETSI). The GMR is derived from the Global System for Mobile Communications (GSM) global cellular standard [1] and has two variants: GMR-1, adopted by Thuraya, TerreStar, and SkyTerra, and GMR-2, adopted by Inmarsat

The official standards exclude ciphers' specifications used in the GMR-1 and GMR-2. A5-GMR-1 and A5-GMR-2, stream ciphers used in GMR-1 and GMR-2, respectively, were disclosed by Driessen et al. in 2012 through reverse engineering [2], [3]. Because satellite communication systems are inherently vulnerable to eavesdropping, the use of robust ciphers is required to ensure data confidentiality. However, several studies have identified the weaknesses of these ciphers and attacks on them.

Except for the polynomials for internal linear feedback shift registers (LFSRs), A5-GMR-1 and the A5/2 stream cipher used in GSM are structurally identical.

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Huang.

TABLE 1. Comparison of previous attacks with ours.

Adversary Model	Technique	Data	Memory	Execution Time	# of Vertex Visit	Brute Force Space	Reference
Known Keystream Attack	Read-Collision Based	$14 \sim 20$	-	-	-	2^{10}	[2]
		$4 \sim 5$	-	-	-	2^{18}	[2]
	Dynamic Guess-and-Determine	1	-	-	-	2^{28}	[8]
	Inversion Attack	1	6 KB	20 ms	-	2^{13}	[9]
		1	2.75 KB	7.2 ms	$2^{19.0}$	$2^{12.8}$	Ours*
	Inversion Attack with Pre-filtration	1	5.25 KB	4.5 ms	$2^{17.5}$	$2^{12.8}$	Ours
		2	5.25 KB	0.62 ms	$2^{12.8}$	$2^{9.2}$	Ours
Ciphertext-only Attack	Inversion Attack with Pre-filtration	2	5.25 KB	1.3 s	$2^{12.8}$	$2^{9.2}$	Ours

The unit of data complexity is frame (15-byte).
Ours*: Our Implementation of the inversion attack of [9].

Briceno disclosed the A5/2 specification through reverse engineering [4]. Goldberg et al. proposed the first attack on A5/2 [5], and Barkan et al. presented a practical ciphertext-only attack [6]. Driessen et al. proposed a ciphertext-only attack on A5-GMR-1 using Barkan et al.'s attack [2], and this attack has recently been significantly improved in terms of time, data, and memory complexity by Lee et al. [7].

A5-GMR-2 has completely different structure from A5/2. Each of the two parts of A5-GMR-2 selects one byte from the session key in every clock. Driessen et al. proposed a known plaintext attack that uses the possibility of colliding these two bytes [2]. Their read-collision based attack reduces the size of the brute-force space from 2^{64} to 2^{10} when 14-20 keystream frames are available.[1] Given 4-5 keystream frames, the brute-force space can be reduced to 2^{18} when the time-data tradeoff is used. Li et al. proposed the dynamic guess-and-determine attack, a low-data complexity attack, which reduces the brute-force space to 2^{28} using one keystream frame [8]. The best publicly known attack on A5-GMR-2 is the inversion attack that Hu et al. presented [9]. The inversion attack has three phases: table generation, dynamic table look-up, and verification. Candidate keys are reduced during table generation and dynamic table look-up phases, and during the verification phase, brute-force search is used to identify the correct key. According to [9], the inversion attack uses one keystream frame and takes approximately 20 milliseconds to recover the session key.

This study examined the GMR-2 system's security and demonstrated a practical ciphertext-only attack on it. Our contributions are twofold. First, we added a new phase called pre-filtration between the table generation and dynamic table look-up phases to enhance the inversion attack of [9]. The pre-filtration phase requires negligible time and accelerates the dynamic table look-up and verification phases. Therefore, our improved attack incorporating the pre-filtration phase takes 4.5 milliseconds and 0.62 milliseconds on average using one and two keystream frames, respectively. This demonstrates that our improved attack is 1.6 and 11.5 times faster than the original inversion attack, respectively, because our implementation of the previous inversion attack takes 7.2 milliseconds to recover the session key with one keystream frame on average.

However, execution time may not provide a reliable comparison because it depends on how the attack is implemented and the environment in which it is run. To demonstrate the improved performance, we present two additional metrics that are independent of the execution environment: the total number of vertex visits because a graph traversal is a primary process for the pre-filtration and dynamic table look-up phases, and the size of the brute-force search space for the verification phase. Table 1 presents the comparison of our and previous works based on these metrics. Our improved attack using one keystream frame requires 2.8 ($2^{1.5}$) times fewer vertex visits than the previous inversion attack. Our improved attack using two keystream frames requires 74 ($2^{6.2}$) times fewer vertex visits and has a brute-force search space that is 12 ($2^{3.6}$) times smaller than the previous inversion attack. Our table generation phase and the previous inversion attack are identical and require negligible time. Additionally, we found that 2.75 KB of memory is sufficient to run the previous inversion attack, while [9] indicated that 6 KB is required. Pre-filtration requires an additional 2.5 KB of memory, resulting in a memory complexity of 5.25 KB for our improved attack.

Second, we present a practical ciphertext-only attack on the GMR-2 system. Our analysis of the GMR-2 standards shows a vulnerability in the CIPHERING MODE COMMAND message type [10], [11], [12], [13]. This message is sent in a 184-bit format, with 173 of the bits being inferable, limiting the degrees of freedom to 2^{11}. According to the standards, the CIPHERING MODE COMMAND message is sent over one of the six channels, but the information provided cannot determine the specific channel used. For each of these six channels, we determined the number of keystream frames that are derived from a single plaintext. This derivation is

[1]The length of one keystream frame is 15 bytes.

typically intuitive, but the diagonal interleaving of GMR-2 makes it unusual. Some frames are associated with multiple plaintexts because the diagonal interleaving mixes frames from different plaintexts during channel coding. According to our analysis, we cannot derive any keystream frames from a single plaintext in two of the six channels. In the remaining four channels, we can derive two or more keystream frames from a single plaintext. If the CIPHERING MODE COMMAND message is sent over one of these four channels, we can perform a ciphertext-only attack by guessing 2^{11} possible plaintexts and conducting our improved known plaintext attack. Our attack outputs the session key only if the attack is conducted using the correctly guessed plaintext; otherwise, it outputs nothing. This enables us to identify the plaintext corresponding to the given ciphertext, which enables a ciphertext-only attack. The required time is equal to the time required to repeat the known plaintext attack 2^{11} times. Therefore, our ciphertext-only attack can recover the session key in 1.3 seconds, thanks to the speed improvement achieved by pre-filtration. The complexity of our ciphertext-only attack is shown in Table 1.

The remainder of this paper is organized as follows. Section II gives the background of the GMR-2 system and some notations required to understand this paper and briefly describes the A5-GMR-2 stream cipher. Section III describes the inversion attack proposed by Hu et al. Section IV presents an improved known plaintext attack. Section V analyzes the relevant standards and presents a practical ciphertext-only attack. Section VI concludes the paper.

II. PRELIMINARIES

A. BACKGROUNDS

This subsection provides some technical background on the GMR-2 system. In GMR-2, the Mobile Earth Station (MES) and network send and receive messages. The MES in GMR-2 is equivalent to the Mobile Station (MS) in GSM [10], which refers to the physical equipment that the subscriber uses to gain access to the telecommunication services offered [14]. Messages are transmitted through different channels according to their purpose and type. Before a message is transmitted, it is formatted, channel-coded, mapped to one or more frames, and then encrypted frame by frame. Throughout this paper, a *plaintext* refers to a formatted message and a *ciphertext* refers to an encrypted frame. Each channel has a different channel coding scheme, and frames from different messages may be mixed during channel coding, which makes the relationship between known plaintext and known keystream frame non-intuitive. A detailed discussion on channel coding and the non-intuitive relationship between known plaintext and known keystream frame is provided in Section V.

The official standards exclude information about the GMR-2's encryption, such as the specifications of the ciphers used. Additionally, the standards do not specify which channels or message types are encrypted. Therefore, our ciphertext-only attack, proposed in Section V, focuses on the CIPHERING MODE COMMAND message type, although there are other message types with similar limited degrees of freedom. This is the only message type that for which standard mentions whether it is encrypted. The standard states that "one of three valid forms of CIPHERING MODE COMMAND is sent in *ciphered mode*" [12]. The details of the CIPEHRING MODE COMMAND message are discussed in Section V.

B. NOTATIONS

We introduce the definitions of terms related to graph theory and several data structures required to understand this paper.

Definition 1 (Reachable [15]): In a directed graph, a vertex v is considered reachable from a vertex u if v can be reached by following the directed edges starting from u. The set of all vertices that are reachable from u is referred to as the reachable set of u.

Definition 2 (Predecessor and Successor [15]): In a directed graph, if a vertex v is reachable from a vertex u, then u is a predecessor of v, and v is a successor of u. If there is a directed edge from u to v, then u is an immediate predecessor of v, and v is an immediate successor of u.

Every vertex in the directed graph is reachable from itself, and a vertex with an outdegree one has a unique immediate successor. Definitions 4 and 5 define stack data structures, queue data structures, and some related operations.

Definition 3 (Stack Data Structure [16]): A stack data structure is a collection of elements in which elements are inserted and removed in a Last-In-First-Out (LIFO) manner. The top of the stack is referred to as the most recently added element. The following three operations can be performed on a stack S:

1) *$S.top$: Returns the value of the element at the top of the stack.*
2) *$S.push(x)$: Adds an element x to the top of the stack.*
3) *$x \leftarrow S.pop$: Removes the element at the top of the stack and assigns its value to the variable x.*

Definition 4 (Queue Data Structure [16]): A queue data structure is a collection of elements in which elements are inserted and removed in a First-In-First-Out (FIFO) manner. The following four operations can be performed on a queue Q:

1) *$Q.front$: Returns the first element in the queue.*
2) *$Q.back$: Returns the last element in the queue.*
3) *$Q.push(x)$: Inserts an element x to the back of the queue.*
4) *$x \leftarrow Q.pop$: Removes the first element of the queue and assigns its value to the variable x.*

C. DESCRIPTION of A5-GMR-2

The specification of A5-GMR-2 stream cipher is disclosed by Driessen et al. through reverse engineering [2]. A5-GMR-2 accepts a 64-bit session key and 22-bit frame number and outputs a 15-byte keystream. It consists of three components: $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$. Fig. 1 shows the overall structure of A5-GMR-2

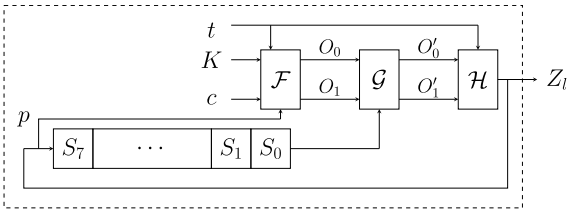

FIGURE 1. Overall structure of A5-GMR-2.

TABLE 2. Definition of τ_1 and τ_2.

x	$\tau_1(x)$	$\tau_2(x)$	x	$\tau_1(x)$	$\tau_2(x)$
0x0	2	4	0x8	3	-
0x1	5	5	0x9	0	-
0x2	0	6	0xA	6	-
0x3	6	7	0xB	1	-
0x4	3	4	0xC	5	-
0x5	7	3	0xD	7	-
0x6	4	2	0xE	4	-
0x7	1	1	0xF	1	-

and Fig. 2, 3, and 4 show the structures of the $\mathcal{F}$-, $\mathcal{G}$-, and $\mathcal{H}$-component, respectively.

The cipher outputs a 1-byte keystream per clock based on internal states. The internal state consists of two 8-byte registers (K and S), a 3-bit counter (c), a 1-bit toggle bit (t), and a 1-byte value (p). When the internal states are ready to generate the keystream, the $\mathcal{F}$-component computes 8-bit O_0 and 4-bit O_1 based on K, c, t, and p. Based on O_0, O_1, and S, $\mathcal{G}$-component computes 6-bit O'_0 and 6-bit O'_1. Finally, $\mathcal{H}$-component generate a 1-byte keystream based on O'_0, O'_1, and t. The detailed process of each component is as follows. The following description assumes that the cipher is at the l-th clock and the notation $(\cdot)_x$ is used to denote a binary value of x bits.

1) $\mathcal{F}$-COMPONENT

In the $\mathcal{F}$-component, two bytes of the session key are used. First, α is calculated from c, p, and t as follows.

$$\alpha = \begin{cases} ((K_c \oplus p)\ \&0 \times 0\mathtt{F})_4 & \text{if } t = 0, \\ (((K_c \oplus p) \gg 4)\ \&0 \times 0\mathtt{F})_4 & \text{if } t = 1. \end{cases} \tag{1}$$

Then, O_0, O_1 are calculated. Two functions τ_1 and τ_2 are defined as Table 2.

$$\begin{cases} O_0 = \left(K_{\tau_1(\alpha)} \ggg \tau_2\left(\tau_1\left(\alpha\right)\right)\right)_8, \\ O_1 = ((((K_c \oplus p) \gg 4) \oplus (K_c \oplus p))\ \&0 \times 0\mathtt{F})_4. \end{cases} \tag{2}$$

2) $\mathcal{G}$-COMPONENT

Three linear transformations $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ are used for the $\mathcal{G}$-component.

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

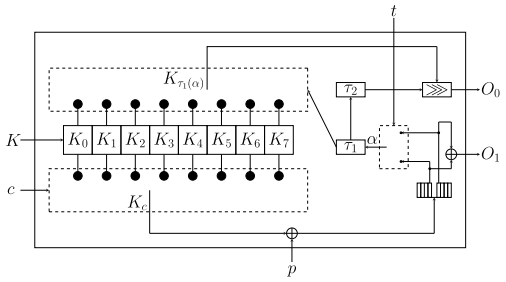

FIGURE 2. $\mathcal{F}$-component.

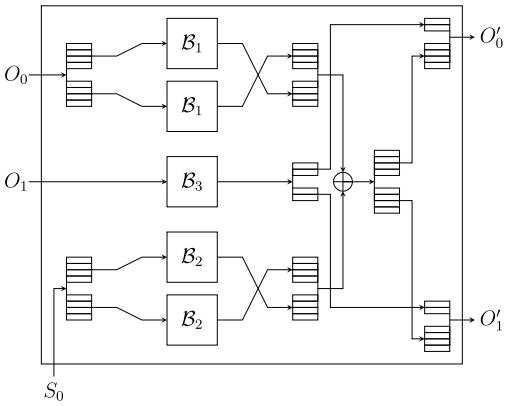

FIGURE 3. $\mathcal{G}$-component.

Fig. 3 illustrates the calculation process for O'_0 and O'_1. We can also briefly express O'_0 and O'_1 as shown in the following equations, where $O_{i,j}$ and $S_{i,j}$ denote the j-th bit of O_i and S_i, respectively.

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

3) $\mathcal{H}$-COMPONENT

The $\mathcal{H}$-component uses two 6-bit-input and 4-bit-output S-boxes, namely $\mathbb{S}_2$ and $\mathbb{S}_6$ that are used in the DES block cipher [17]. A detailed description of $\mathbb{S}_2$ and $\mathbb{S}_6$ can be found in Table 3. Note that all values in the table are presented in hexadecimal format.

Unlike DES, in A5-GMR-2, the most-significant 4-bits determine the column index and the least-significant 2-bits determine the row index. Based on the toggle-bit t, 1-byte keystream at l-th clock, Z_l, is calculated as follows.

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

4) INITIALIZATION PHASE

The internal states of A5-GMR-2 are initialized as follows:

TABLE 3. Description for $\mathbb{S}_2$ and $\mathbb{S}_6$.

$\mathbb{S}_2$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	F	1	8	E	6	B	3	4	9	7	2	D	C	0	5	A
1	3	D	4	7	F	2	8	E	C	0	1	B	6	9	B	5
2	0	E	7	B	A	4	D	1	5	8	C	6	9	3	2	F
3	D	8	A	1	3	F	4	2	B	6	7	C	0	5	E	9

$\mathbb{S}_6$

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	C	1	A	F	9	2	6	8	0	D	3	4	E	7	5	B
1	A	F	4	2	7	C	9	5	6	1	D	E	0	B	3	8
2	9	E	F	5	2	8	C	3	7	0	4	A	1	D	B	6
3	4	3	2	C	9	5	F	A	B	E	1	7	6	0	8	D

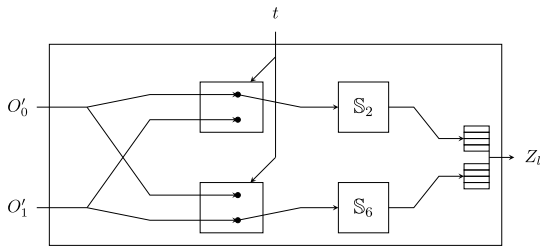

FIGURE 4. $\mathcal{H}$-component.

- c, t, and p are set to zero.
- 64-bit session key is written into the 8-byte register K in the $\mathcal{F}$-component.
- 8-byte register S is initialized with a 22-bit frame number.[2]

5) GENERATION PHASE

Once the internal states have been initialized, the cipher is clocked to generate a keystream. Following each clock cycle, the cipher updates its internal states as follows:

- The cipher generates 1-byte Z_l based on the current internal states.
- $c \leftarrow (c+1) \bmod 8$, $t \leftarrow t \oplus 1$, and $p = Z_l$.
- $S_i \leftarrow S_{i+1}$ for $i = 0, 1, \ldots, 6$ and $S_7 \leftarrow Z_l$.

The cipher is clocked 23 times, the first 8 bytes are discarded, and the next 15 bytes are used as a keystream. In the rest of this paper, we count the index of the clock after the first 8 clocks and denote the 15-byte keystream as $Z = (Z_0, Z_1, .., Z_{14})$.

III. INVERSION ATTACK ON A5-GMR-2

The inversion attack, proposed by Hu et al. [9], is currently the best publicly known attack on the A5-GMR-2 cipher. This attack was named an inversion attack because it uses the inverse properties of the $\mathcal{F}$-, $\mathcal{G}$-, and $\mathcal{H}$-component of A5-GMR-2. The phrase *inverse property* in this context refers to a property that allows the input to be inferred from the output.

[2]We omit explaining the detailed process because it is irrelevant to our attack.

A. INVERSE PROPERTY of A5-GMR-2

This subsection introduces and combines the inverse properties of each component.

1) INVERSE PROPERTY OF $\mathcal{H}$-COMPONENT

Each hexadecimal digit from 0×0 to $0\text{x}\mathtt{F}$, appears exactly once in each row of Table 3. Therefore, given Z_l and l,[3] there are four possibilities for O'_0 and for O'_1, that is, there are 16 possibilities for (O'_0, O'_1).

2) INVERSE PROPERTY OF $\mathcal{G}$-COMPONENT

Equation (3) can be re-written as follows:

$$\begin{pmatrix} O'_{0,5} \\ O'_{0,4} \\ O'_{0,3} \\ O'_{0,2} \\ O'_{0,1} \\ O'_{0,0} \\ O'_{1,5} \\ O'_{1,4} \\ O'_{1,3} \\ O'_{1,2} \\ O'_{1,1} \\ O'_{1,0} \end{pmatrix} = \begin{pmatrix} 1&0&0&1&0&0&0&0&0&0&0&0 \\ 1&1&0&1&0&0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&1&1 \\ 0&0&0&0&0&0&0&0&1&0&0&1 \\ 0&0&0&0&1&0&0&1&0&0&0&0 \\ 0&0&0&0&1&1&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \end{pmatrix} \cdot \begin{pmatrix} O_{0,7} \\ O_{0,6} \\ O_{0,5} \\ O_{0,4} \\ O_{0,3} \\ O_{0,2} \\ O_{0,1} \\ O_{0,0} \\ O_{1,3} \\ O_{1,2} \\ O_{1,1} \\ O_{1,0} \end{pmatrix} \oplus \begin{pmatrix} S_{0,5} \\ S_{0,7} \\ S_{0,4} \\ S_{0,6} \\ 0 \\ 0 \\ S_{0,1} \\ S_{0,3} \\ S_{0,0} \\ S_{0,2} \\ 0 \\ 0 \end{pmatrix} \tag{4}$$

As the 12×12 binary matrix in Eq. (4) is invertible, given (O'_0, O'_1) and S_0, (O_0, O_1) can be uniquely determined.

3) INVERSE PROPERTY OF $\mathcal{F}$-COMPONENT

The following four equations can be derived from Eq. (1).

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

Using Eq. (5), we can derive 16 possible values for K_c, given O_1 and p. Specifically, if $O_{1,i} \oplus p_{i+4} \oplus p_i = 0$, then $(K_{c,i+4}, K_{c,i})$ can be either (0, 0) or (1, 1), while $(K_{c,i+4}, K_{c,i})$ can be either (1, 0) or (1, 0) otherwise. Using Eq. (1), we can then calculate α for each possible value of K_c. Finally, Eq. (2) allows us to compute $(K_{\tau_1(\alpha)}, \tau_1(\alpha))$ given O_0 and α. Therefore, for any given values of O_0, O_1, and p, there are 16 possible combinations of $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$ that satisfy the equations.

4) COMBINING INVERSE PROPERTIES OF $\mathcal{F}$, $\mathcal{G}$, AND $\mathcal{H}$

Given l, Z_l, S_0, and p, we can combine the inverse properties of the three components to find 256 possibilities for $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$. The Proposition 1 states that only one of these possibilities is compatible with the session key.

Proposition 1 [9]: Given l, Z_l, S_0, and p, we can find 256 possibilities for $\left(K_{l-8}, K_{\tau_1(\alpha)}, \tau_1(\alpha)\right)$. Let (x_0, y_0, z_0), (x_1, y_1, z_1), $\cdots$, $(x_{255}, y_{255}, z_{255})$ be the possibilities. Then,

[3]Note that t can be naturally derived from l as $t = l \bmod 2$.

all $x_0, \cdots, x_{255}$ *are distinct, meaning that* $\{x_0, x_1, \cdots, x_{255}\} = \{0, 1, \cdots, 255\}$.

Therefore, we can express the possibility (x, y, z) for $(K_c, K_{\tau_1(\alpha)}, \tau_1(\alpha))$ as follows: if $K_c = x$, then $K_z = y$.

B. ATTACK PROCEDURE

Although our description of the inversion attack may slightly differ from that of [9], the underlying principle and attack complexity remain the same. These modifications aim to make our improved attack more understandable, which is described in Section IV.

The inversion attack is divided into three phases: table generation, dynamic table look-up, and verification. The attack uses one keystream frame (15-byte) to recover the session key. We denote the 15-byte known keystream as $Z = (Z_0, Z_1, \ldots, Z_{14})$.

1) PHASE 1: TABLE GENERATION

For $8 \leq l \leq 14$, we can find 256 possibilities for $(K_{l-8}, K_{\tau_1(\alpha)}, \tau_1(\alpha))$ from l, Z_l, $S_0\,(= Z_{l-8})$, and $p\,(= Z_{l-1})$ based on Proposition 1. From 7×256 possibilities we construct a directed graph G as follows. Every vertex in the resulting graph G has an outdegree of one.

1) G has 8×256 vertices. We denote each vertex as $v_{i,j}$, where $0 \leq i \leq 7$, $0 \leq j \leq 255$, which corresponds to "$K_i = j$."
2) If there is a possibility (x, y, z) for (K_i, K_j, j), we add a directed edge from vertex $v_{i,x}$ to vertex $v_{z,y}$ to represent the condition "if $K_i = x$, then $K_z = y$".
3) For each vertex $v_{7,j}$, where $0 \leq j \leq 255$, we add a self-loop on it.

2) PHASE 2: DYNAMIC TABLE LOOK-UP

Assume that there exist directed edges from v_{i_0,j_0} to v_{i_1,j_1} and from v_{i_1,j_1} to v_{i_2,j_2}. The existence of these two directed edges implies that "if $K_{i_0} = j_0$ then $K_{i_1} = j_1$" and "if $K_{i_1} = j_1$ then $K_{i_2} = j_2$", respectively, which can be simplified to "if $K_{i_0} = j_0$ then $K_{i_1} = j_1$ and $K_{i_2} = j_2$." From this concept, we present Proposition 2, which generalizes the idea of combining multiple relations.

Proposition 2: Let G be a directed graph constructed in the table generation phase. Assume that $v_{x,y}$ *is reachable from* $v_{i,j}$ *in G. This implies that "if* $K_i = j$ *then* $K_x = y$."

Assume that $v_{i,j'}$ is reachable from $v_{i,j}$, where $j \neq j'$. Based on Proposition 2, this implies that "if $K_i = j$ then $K_i = j'$" which is a contradiction. This leads us to the conclusion that $K_i \neq j$. Proposition 3 generalizes this idea.

Proposition 3: Let G be a directed graph constructed in the table generation phase. If a reachable set of vertex $v_{i,j}$ *of G includes two vertices* $v_{x,y}$ *and* $v_{x,y'}$ *such that* $y \neq y'$, *then* $v_{i,j}$ *can never be compatible with the correct session key, meaning* $K_i \neq j$.

All session key candidates can be mapped to $\{v_{0,j_0}, v_{1,j_1}, \cdots, v_{7,j_7}\}$, where $0 \leq j_0, j_1, \cdots, j_7 \leq 255$. If there exists j_i, such that the reachable set of v_{i,j_i} includes $v_{x,y}$, where $y \neq j_x$, then $\{v_{0,j_0}, v_{1,j_1}, \cdots, v_{7,j_7}\}$ never be the correct session key. Therefore, we can restrict the session key candidates to the sets of vertices that satisfy the following: Assume that $\mathcal{S} = \{v_{0,j_0}, v_{1,j_1}, \cdots, v_{7,j_7}\}$, and $\mathcal{R}_{0,j_0}, \mathcal{R}_{1,j_1}, \cdots, \mathcal{R}_{7,j_7}$ are the reachable sets of $v_{0,j_0}, v_{1,j_1}, \cdots, v_{7,j_7}$, then $\mathcal{S} = \mathcal{R}_{0,j_0} \cup \mathcal{R}_{1,j_1} \cup \cdots \cup \mathcal{R}_{7,j_7}$. Algorithm 1 gives the process of finding all such $\mathcal{S}$ in G and storing them in KC, a set of key candidates. We begin at the vertex $v_{0,0}$ and traverse the graph G by following its directed edges in a deterministic manner[4] (Steps 13-14 and 24-26). We store the visited vertices in a stack data structure and perform operations as follows:

- If a contradiction occurs, the algorithm backtracks to a new starting point (Steps 1-11 and 15-16).

[4]every vertex in G has outdegree one.

Algorithm 1 Dynamic Table Look-Up

Input: G, a directed graph constructed in the table generation phase
Output: KC, a set of session key candidates

```
1  Function BackTracking(S, v_{i,j}):
2      while S.top is immediate predecessor of v_{i,j} do
3          v_{i,j} ← S.pop
4      if j = 255 then
5          if i = 0 then
6              return ({}, Null, False)
7          v_{i,j} ← S.pop
8          Go to Step 2.
9      else
10         S.push(v_{i,j+1})
11         return (S, v_{i,j+1}, True)

12 Function Main:
13     S ← Empty Stack, v_cur ← v_{0,0}, S.push(v_cur)
14     v_{i,j} ← immediate successor of v_cur
15     if there exist v_{i,j'} ∈ S such that j ≠ j' then
16         (S, v_cur, res) ← BackTracking(S, v_{i,j})
17     else if v_{i,j} ∈ S and |S| = 8 then
18         KC ← KC ∪ {S}
19         (S, v_cur, res) ← BackTracking(S, v_{i,j})
20     else if v_{i,j} ∈ S and |S| ≠ 8 then
21         i ← min i | ∄v_{i,j} ∈ S
22         v_cur ← v_{i,0}, S.push(v_cur)
23         res ← True
24     else
25         v_cur ← v_{i,j}, S.push(v_cur)
26         res ← True
27     if res = True then
28         Go to Step 14.
29     return KC
```

- If the desired set is found, the set is stored and then the algorithm backtracks to a new starting point (Steps 1-11 and 17-19).
- If the number of vertices visited through traversing is less than eight, the algorithm jumps to a new starting point (Steps 20-23).
- The algorithm terminates when there are no vertices to backtrack. (Steps 27-29).

3) PHASE 3: VERIFICATION

During the verification phase, we find the correct one among the session key candidates in KC that is obtained during the dynamic table look-up phase. We pick a key candidate and check if the cipher generates the given keystream with the picked key candidate. According to the previous work [9], only one of the key candidates generates the known keystream with a 97.2% probability. Otherwise (worst case), multiple candidates generate the given keystream and the adversary cannot uniquely determine the correct session key. Previous work stated that even in the worst-case scenario, the adversary can determine the correct session key if only one additional keystream byte is provided. However, this argument is flawed because, in GMR-2, the size of the frame is fixed to 15 bytes, that is, the cipher never generates the 16th keystream byte. Instead, to determine the correct session key in the worst-case scenario, the adversary needs one more keystream frame.

C. COMPLEXITY ANALYSIS

Hu et al. stated that the inversion attack recovers the session key in an average of 20 milliseconds and requires 6 KB memory [9]. For a fair comparison with our improved attack discussed in Section IV, we implement the inversion attack directly on our computer. Our implementation takes 7.2 milliseconds to recover the session key.[5] The only memory required is for storing the directed graph G constructed in the table generation phase. For $v_{i,j}$ in G, assume that $v_{x,y}$ is an immediate successor of $v_{i,j}$. We store x and y using i and j as indices. Because we need 3-bit and 8-bit to store x and y, respectively, our implementation only requires 2.75 KB ($=2048 \times 11$ bits) of memory.

IV. IMPROVED KNOWN PLAINTEXT ATTACK

We introduce a new phase called the *pre-filtration* phase and add it between the table generation and dynamic table look-up phases to improve the time complexity of the inversion attack. The pre-filtration phase requires negligible time, but significantly reduces the time required for the dynamic table lookup and verification phases, which results in an overall time complexity reduction.

A. NEW PHASE: PRE-FILTRATION

First, we define the *black* vertex and *dead* vertex as Definition 5 and the *reachable index set* as Definition 6.

[5]We use our implementation of the previous inversion attack for comparison with our improved attack. Additionally, we present two more metrics other than execution time to demonstrate improvement in Section IV.

Definition 5 (Black Vertex and Dead Vertex): Let G be a directed graph constructed in the table generation phase. For a vertex v in G, v is called a dead vertex if its reachable set includes any pair of vertices $(v_{i,j},v_{i,j'})$ where $j \neq j'$. v is called a black vertex if it is not a dead vertex.

Definition 6 (Reachable Index Set): Let G be a directed graph constructed in the table generation phase. For a vertex v in G, let $\mathcal{R}$ be a reachable set of v. Then, we define the reachable index set of v as $\mathcal{R}_{idx} = \{i : \exists j \ s.t. \ v_{i,j} \in \mathcal{R}\}$.

According to Proposition 3, a dead vertex can never be an element of the set we are looking for in the dynamic table look-up phase. Therefore, traversing a dead vertex in the dynamic table look-up phase is futile. We effectively find all dead vertices and exclude them from G to reduce the time complexity of the dynamic table look-up phase.

Algorithm 2 gives the process of classifying all vertices in graph G as either black or dead. We traverse the graph G following the directed edges and use a queue, Q, to store the path of the visited vertices. For simplicity of explanation, we call a vertex that we do not yet know whether it is a black or dead vertex a white vertex, and a gray vertex if it is in Q.

We initially set all 2048 vertices to white and operate based on the following five rules: Rule 1 outlines the action to take when the queue Q is empty, and Rules 2-5 explain the operation for the remaining cases. In Rules 2-5, v_{cur} refers to the last vertex pushed to Q and $v_{i,j}$ refers to the immediate successor of v_{cur} (Steps 3,6, and 11).

- **Rule 1:** Push any white vertex to Q if Q is empty. If there are no white vertices left, Algorithm 2 ends. (Steps 2-5)
- **Rule 2:** If $v_{i,j}$ is a white vertex, pop vertices while there exist $v_{i,j'}$ inside Q. According to Proposition 3, all popped vertices are dead vertices. Then, push $v_{i,j}$ to Q. (Steps 7-12)
- **Rule 3:** If $v_{i,j}$ is a gray vertex, this implies that all the vertices in Q are black vertices. As we pop vertices one by one from Q, we store the reachable index set of each vertex. Note that a cycle is formed from $v_{i,j}$. (Steps 13-17)
- **Rule 4:** If $v_{i,j}$ is a black vertex, let $\mathcal{R}_{idx}$ be its reachable index set. Suppose that there exist x and y such that $v_{x,y}$ is in Q and x is in $\mathcal{R}_{idx}$. Then, there must exist a y' such that the vertex $v_{x,y'}$ is reachable from $v_{i,j}$. Given that $v_{i,j}$ is black, we know that $v_{x,y'}$ is also black. Additionally, because $v_{x,y}$ is gray, we must have $y \neq y'$. This causes a contradiction. We pop vertices while such $v_{x,y}$ exist in Q and classify them as dead. The remaining vertices in Q are black. As we pop the remaining vertices one by one from Q, we store the reachable index set of each vertex. (Steps 18-26)
- **Rule 5:** If $v_{i,j}$ is a dead vertex, all vertices in Q are dead vertices. Pop all the vertices from Q. (Steps 27-30)

Only white vertices are pushed to Q, and when a vertex is popped out of Q, it is classified as either black or dead. That is, pre-filtration visits all vertices only once.

We now explain how to extend the pre-filtration phase given two keystream frames. We conduct the table generation

Algorithm 2 Pre-Filtration

Input: G, a directed graph constructed in the table generation phase

$Q \leftarrow$ Empty Queue
if $\exists$ white vertex $v \in G$ **then**
  $v_{cur} \leftarrow v$, $Q.push(v_{cur})$
else
  Terminate Algorithm

$v_{i,j} \leftarrow$ immediate predecessor of v_{cur}
if $v_{i,j}$ is white **then**
  while $\exists v_{i,j'} \in Q$ *such that* $j \neq j'$ **do**
    $v \leftarrow Q.pop$
    Classify v as dead.
  $v_{cur} \leftarrow v_{i,j}$, $Q.push(v_{cur})$
  Go to Step 7.
else if $v_{i,j}$ is gray **then**
  while Q is not empty **do**
    $v \leftarrow Q.pop$
    Classify v as black
    Store the reachable index set of v.
else if $v_{i,j}$ is black **then**
  Let $\mathcal{R}_{idx}$ be a reachable index set of $v_{i,j}$.
  while $\exists x, y\ s.t.\ v_{x,y} \in Q$ *and* $x \in \mathcal{R}_{idx}$ **do**
    $v \leftarrow Q.pop$
    Classify v as dead.
  while Q is not empty **do**
    $v \leftarrow Q.pop$
    Classify v as black
    Store the reachable index set of v.
else
  while Q is not empty **do**
    $v \leftarrow Q.pop$
    Classify v as dead
Go to Step 2.

and pre-filtration phase with the first known keystream frame. Then, we independently conduct the table generation phase with the second known keystream frame. Let the resulting graph be G. Before starting the pre-filtration phase on G, we initialize the vertices as follows: If a vertex $v_{i,j}$ is determined to be a black vertex after the pre-filtration phase with the first keystream frame, we initialize $v_{i,j}$ to a white vertex. Otherwise, it is initialized as a dead vertex. Following the second pre-filtration, far fewer vertices are classified as black, which significantly reduces the time required for the dynamic table look-up phase. Additionally, the size of the KC is also reduced, which significantly reduces the time required for the verification phase.

B. COMPLEXITY ANALYSIS

Table 1 lists the results of testing the inversion attack and our attack 10,000 times each. For a fair comparison, we present not only the average execution time but also two additional metrics that are not influenced by the execution environment. The first metric is the number of vertex visits during the pre-filtration and dynamic table look-up phases. Because traversing the graph is the main process for both the pre-filtration and dynamic table look-up phases, it is a fair metric to validate the speedup of the dynamic table look-up phase. The second metric is the size of the brute-force space (the size of KC), which allows a direct comparison of the time required for the verification phase.

Our implementation of the previous inversion attack takes 7.2 milliseconds to recover the session key, requires 2^{19} vertex visits for the dynamic table look-up phase, and has the brute-force space size of $2^{12.8}$. Our improved inversion attack with one keystream frame takes 4.5 milliseconds to recover the session key, requires $2^{17.5}$ vertex visits for the pre-filtration and dynamic table look-up phase, and has the brute-force space size of $2^{12.8}$. Our improved inversion attack with two keystream frames takes 0.62 milliseconds to recover the session key, requires $2^{12.8}$ vertex visits for the pre-filtration and dynamic table look-up phase, and has the brute-force space size of $2^{9.2}$. The results are summarized in Table 1. The experimental results demonstrate that, compared with the previous inversion attack, our improved attacks are between 1.6 and 11.5 times better in terms of time complexity. To store the reachable index set and state (white, gray, black, or dead) of each vertex, 8-bit and 2-bit are required, respectively. Therefore, the memory complexity of our attack is 5.25 KB (=2.75 KB + 2048 × 10 bits).

V. CIPHERTEXT-ONLY ATTACK ON GMR-2

This section presents a practical ciphertext-only attack on the GMR-2 system. We identify a vulnerability in the GMR-2 system by analyzing the relevant standards [11], [12] and develop a ciphertext-only attack based on it. Our attack specifically targets the CIPHERING MODE COMMAND message. We limit the degrees of freedom of this message type to 2^{11} by inferring all but 11 bits. We also find the six channels through which the CIPHERING MODE COMMAND message might be transmitted. We analyze these six channels and demonstrate that if the four specific channels among them are used to transmit CIPHERING MODE COMMAND messages, the session key can be recovered through a ciphertext-only attack.

A. INFERRING CIPHERING MODE COMMAND MESSAGE

The GMR-2 standard provides a secure satellite communication system; however, our examination of the standards reveals a vulnerability that allows for the inference of certain messages, specifically those of type CIPHERING MODE COMMAND. This subsection presents the process of inferring all but 11 bits of the CIPHERING MODE COMMAND plaintext based on two GMR-2 standards [11] and [12].

First, we identify the key clauses in [12]. Clause 10.1.9 provides the functional definitions and contents of the CIPHERING MODE COMMAND message. This message is sent on the main Satellite Dedicated Control Channel (S-DCCH) from the network to the MES, to indicate whether ciphering will be performed or not. The contents are three bytes long as shown in Fig. 5. Clause 4.4.7.2 defines three valid CIPHERING MODE COMMAND message formats, but we concentrate on only one transmitted in *ciphered mode*. This form indicates no ciphering and is received by the MES in ciphered mode. The other two forms are transmitted in *not ciphered mode*. Therefore, we can deduce that all encrypted CIPHERING MODE COMMAND messages indicate no ciphering.

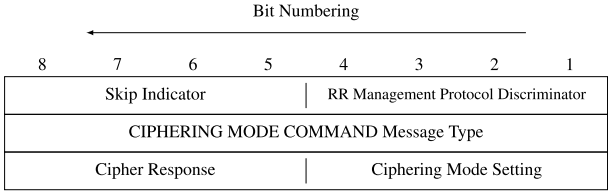

FIGURE 5. Contents of CIPHERING MODE COMMAND message.

The value of contents can be inferred from Clause 11.[6] According to Clause 11.2, the protocol discriminator for the radio resource (RR) management message is represented by the 4-bit value of 0110. Clause 11.3 states that any message received with a skip indicator other than 0000 should be ignored. The 8-bit message type for the CIPHERING MODE COMMAND message is defined in Clause 11.4 as 00110101. According to Clauses 11.5.2.9 and 11.5.2.10, the configuration of the 1 byte for ciphering mode setting and cipher response is shown in Fig. 6. In the case of the message indicating *no ciphering*, the 1-bit SC is set to 1. The 3-bit algorithm indicator is set to 000 if SC is 1. The 3-bit spare is always set to 000. The CR, a 1-bit value, is used to indicate information that should be included in the response to the CIPHERING MODE COMMAND message, and we cannot infer its value.

We now examine the key clauses in [11] to understand the content formatting and deduce the remaining bits. According to Clause 5.1, the frame format used for the CIPHERING MODE COMMAND message is as shown in Fig. 7.[7] Clauses 5.3, 5.4, and 5.5 specify that the address field, control field, and length indicator field are each 1 byte long. Clause 8.8.3 details the maximum length of the information field for each channel, with a 20-byte limit for the main S-DCCH. The total length of the information field and the fill bits is equal to the maximum length of the information field. If the content length is less than the maximum length of the information field, the remaining space is allocated to the fill bits. Because the contents of the CIPHERING MODE COMMAND message are three bytes long, the information

[6]The detailed use of each element, which is defined in [18], will not be covered because it falls outside the scope of this paper.

[7]The length of each field shown in 7 is specific to the CIPHERING MODE COMMAND message and may differ for other message types.

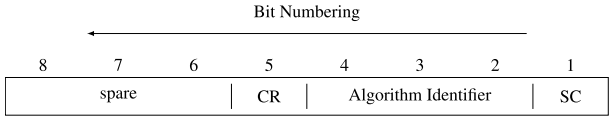

FIGURE 6. Configuration of Ciphering mode setting and Cipher response.

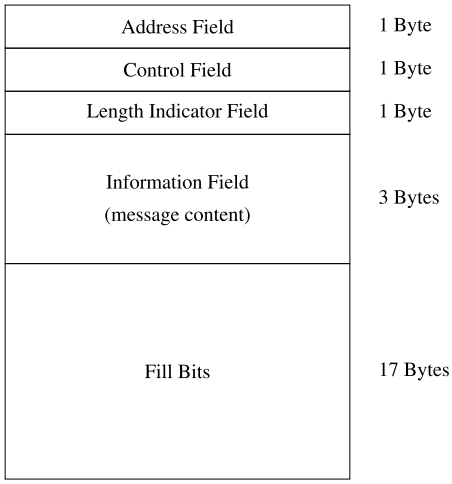

FIGURE 7. Frame format for CIPHERING MODE COMMAND message.

field is 3 bytes long, and the remaining 17 bytes are allocated to the fill bits.

Clause 5.2 specifies that all bytes of the fill bits in messages sent by the network must be set to the binary value 00101011. The formats of other fields are described in Clause 6. Clause 6.1 states that the spare bits are set to 0. Clause 6.2 outlines the address field's format and is shown in Fig. 8. Clause 6.2 also describes the 2-bit link protocol discriminator (LPD), but we cannot determine its value from the information provided. Further details regarding the variables are outlined in Clause 6.3. The address field extension bit (EA) is used to manage situations in which the length of the address field is extended. Because the address field in our target message is 1 byte long, EA is set to 1. The command/response field (C/R) bit indicates whether a frame is a command or a response. Our target message is a command from the network to the MES; therefore, C/R is set to 1. Clause 6.3.3 states that the service access point identifier (SAPI) for radio resource management messages is set to 000.

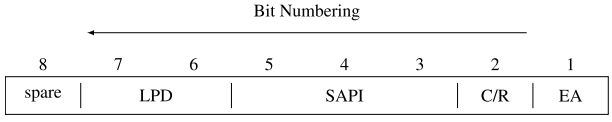

FIGURE 8. Address field format.

Clause 6.4 describes the format of the control field, but we cannot determine its value from the information provided. Clause 6.6 details the format of the length indicator field, as shown in Fig. 9. The length indicator field extension (EL) bit is used to manage situations in which the length of the length indicator field is extended. Because the length indicator field in our target message is 1 byte long, EL is set to 1.

The more data bit (M) is used to indicate the segmentation of frames. The information field in our target message is set to 3 bytes long, which is less than the maximum length of 20 bytes. Therefore, segmentation is not required, and M is set to 0. The length indicator (L) indicates the length of the information field. Because the information field in our target message is 3 bytes long, L takes the binary value 000011.

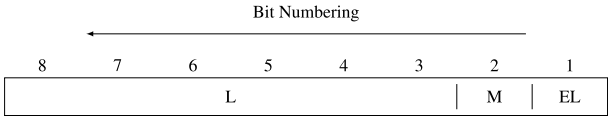

FIGURE 9. Length indicator field format.

Fig. 10 shows the matching of all inferred bits to the format, where '−' indicates the bits we could not infer. Note that the values are written in order from higher bit numbering according to Clause 5.8.3 of [11].

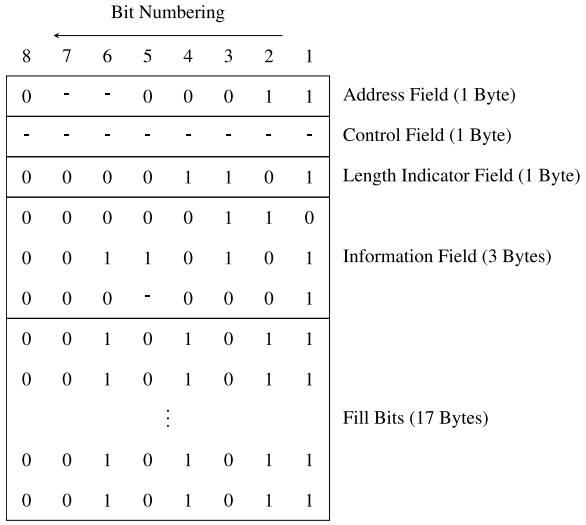

FIGURE 10. Inferred Bits of CIPHERING MODE COMMAND message.

B. ANALYZING THE MAIN S-DCCH IN GMR-2

In the previous subsection, we discovered that the CIPHERING MODE COMMAND is sent on the main S-DCCH. Clause 3.1 of [12] states that S-FACCH or S-SDCCH is called the main S-DCCH in GMR-2. According to the standard [13], 4 S-FACCHs (S-FACCH/Q2.4, H2.4, QBS, HRS) and 2 S-SDCCHs (S-SDCCH/E, Q) are supported in the GMR-2 system. We analyze how a plaintext corresponds to a keystream in each of these six channels.

We use S-FACCH/HR2.4 (Satellite Fast S-TCH/HR2.4 Associated Control Channel) as an example. In S-FACCH/HR2.4, a plaintext is channel coded as follows: A plaintext of 184 bits is first extended with the 40-bit fire code and 24 zeros, resulting in a block of 248 bits. This block is then encoded using a 1/4, 64-state convolutional code, and 32 coded bits are punctured, leading to a length of 960 bits, represented as $c = \{c(0), c(1), \cdots c(959)\}$.[8] Then, c is split into two 480 bit blocks, ce and co as follows:

$$ce(k) = c(2k),\ co(k) = c(2k+1) \text{ for } k = 0, 1, \cdots, 479.$$

ce is then divided into four equal parts, each consisting of 120 bits. The first part, SubGroup_1, consists of $c(0)$ to $c(119)$; the second part, SubGroup_2, consists of $c(120)$ to $c(239)$; and so on. We denote SubGroup_i as $ce_i = \{ce_i(0), \cdots ce_i(119)\}$. Then, 15 SubGroups, SubGroup_{-6}, to SubGroup_8, are *diagonally interleaved* together. Note that for $i \leq 0$, SubGroup_i represents 120 coded bits from the previous data, and for $i > 4$, SubGroup_i represents 120 coded bits from the subsequent data. Set it to 120 zero bits, if there is no such data. Among the diagonally interleaved blocks, there are 11 blocks related to the current data. We denote them from $\text{SubGroup}'_1$ to $\text{SubGroup}'_{11}$ and they are as follows:

$$\text{SubGroup}_i : ce'_i(0), \cdots, ce'_i(119),$$
$$ce'_i(j) = ce_{i-7+(j \bmod 8)}(j)$$

Each $\text{SubGroup}'_i$ is further interleaved with a 12×10 block interleaver. We denote the block-interleaved $\text{SubGroup}'_i$ as ie_i and the blocks obtained through the same process with co as io_i. Then, ie_i and io_i are alternately mapped onto consecutive frames.

Note that 22 frames are required to transmit a single plaintext. Fig. 11 shows the distribution of bits derived from a single plaintext over 22 frames. A complete keystream frame cannot be derived from a single plaintext in S-FACCH/HR2.4; at least two plaintexts are required. Additionally, we can derive two complete keystream frames from two plaintexts. Similar to S-FACCH/HR2.4, the remaining five channels can also be analyzed. Fig. 11 shows the distribution of bits derived from a single plaintext for each of the remaining five channels. Table 4 summarizes how a

TABLE 4. Relationship between plaintext and keystream in the six channels through which CIPHERING MODE COMMAND might be transmitted.

Channel Type	Channel Name	$\#F$	$\#P1$	$\#P2$
Standalone Dedicated Control Channel	S-SDCCH/E S-SDCCH/Q	4	1	
Fast Associated Control Channel	S-FACCH/HR2.4	0	2	
	S-FACCH/Q2.4	0	2	3
	S-FACCH/QBS	2	1	
	S-FACCH/HRS	2	1	

$\#F$: # of complete keystream frame derived from a single plaintext
$\#P1$: Min # of plaintexts to derive one complete keystream frame
$\#P2$: Min # of plaintexts to derive two complete keystream frame

[8] The detailed process of generating c can be found in [13] and is irrelevant for this paper. Only the fact that c is derived from a single plaintext is significant.

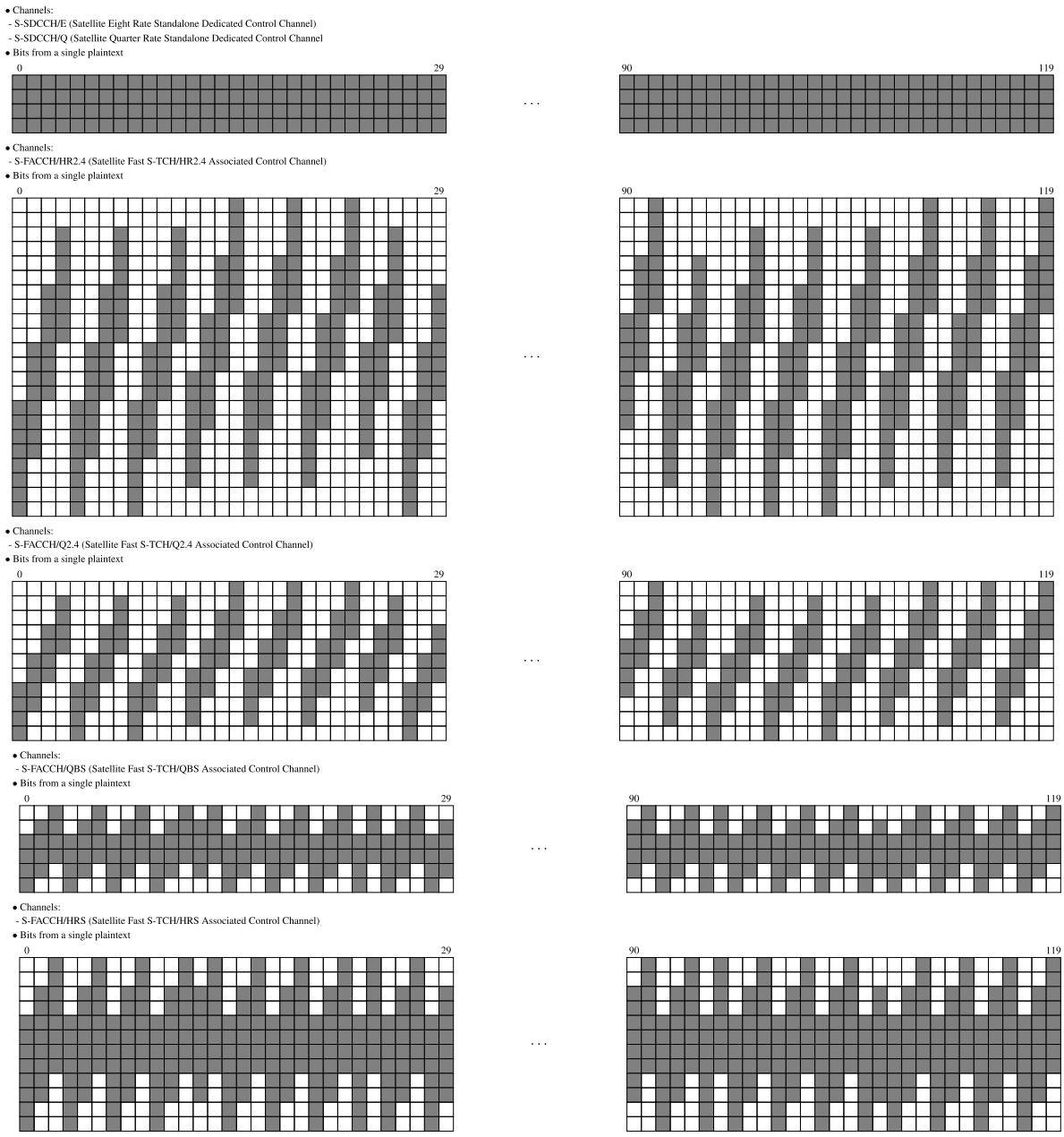

FIGURE 11. Distribution of bits from a single plaintext in six channels through which CIPHERING MODE COMMAND message might be transmitted.

known plaintext corresponds to a known keystream for each channel.

C. SESSION KEY RECOVERY WITH INFERRED MESSAGE

We can infer all but 11 bits of the CIPHERING MODE COMMAND message from the previous subsection. We also know that, given a single plaintext, we can derive at least two complete keystream frames in S-SDCCH/E, S-SDCCH/Q, S-FACCH/QBS, and S-FACCH/HRS.

If one of these four channels is used to transmit the CIPHERING MODE COMMAND message, we can perform a ciphertext-only attack by guessing 2^{11} possible plaintexts, computing the corresponding keystream frames, and conducting our improved known plaintext attack with those keystream frames. Our attack only outputs the session key if the attack is conducted with the keystream from the correctly guessed message; otherwise, it outputs nothing. This allows us to distinguish the message corresponding to the given ciphertext and recover the session key. The required time is approximately 1.3 seconds, which is equal to the time required to repeat the known plaintext attack 2^{11} times.

VI. CONCLUSION

In this study, we analyzed the security of the GMR-2 satellite communication standard. First, we proposed an improved inversion attack on A5-GMR-2. Our attack,

using pre-filtration, recovers the session key within 4.5 milliseconds given one keystream frame and within 0.62 milliseconds given two keystream frames. Additionally, we investigated the relationship between known plaintext and keystream frames in each channel of GMR-2. Our analysis revealed that in some channels, two or more keystream frames are derived from a single plaintext, making the speedup of our attack given two keystream frames more significant. Furthermore, we discovered that GMR-2 uses an inferrable message type. By combining our inversion attack with pre-filtration and the presence of inferrable message types, we presented a practical-time ciphertext-only attack on GMR-2 systems for the first time.

REFERENCES

[1] *GEO-Mobile Radio Interface Specifications; Part 1: General Specifications; Sub—Part 2: Introduction to the GMR-2 Family of Specifications; GMR-2 01.201*, Standard TS 101 377-1-2, European Telecommunications Standards Institute (ETSI), 2001.
[2] B. Driessen, R. Hund, C. Willems, C. Paar, and T. Holz, "Don't trust satellite phones: A security analysis of two satphone standards," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 128–142.
[3] B. Driessen, "Practical cryptanalysis of real-world systems," Ph.D. dissertation, Dept. Elect. Eng., Ruhr-Univ., Bochum, Germany, 2013.
[4] M. Briceno, I. Goldberg, and D. Wagner. (1999). *A Pedagogical Implementation of the GSM A5/1 and A5/2*. [Online]. Available: http://www.scard.org
[5] I. Goldberg, D. Wagner, and L. Green, "The (real-time) cryptanalysis of A5/2," Tech. Rep., 1999.
[6] E. Barkan, E. Biham, and N. Keller, "Instant ciphertext-only cryptanalysis of GSM encrypted communication," in *Proc. 23rd Annu. Int. Cryptol. Conf.*, vol. 2729. Santa Barbara, CA, USA, Aug. 2003, pp. 600–616. [Online]. Available: https://iacr.org/archive/crypto2003/27290598/27290598.pdf
[7] D. Lee, D. Hong, J. Sung, S. Kim, and S. Hong, "Improved ciphertext-only attack on GMR-1," *IEEE Access*, vol. 10, pp. 1979–1989, 2022, doi: 10.1109/ACCESS.2021.3139614.
[8] R. Li, H. Li, C. Li, and B. Sun, "A low data complexity attack on the GMR-2 cipher used in the satellite phones," in *Fast Software Encryption*. Berlin, Germany: Springer, 2013, doi: 10.1007/978-3-662-43933-3_25.
[9] J. Hu, R. Li, and C. Tang, "A real-time inversion attack on the GMR-2 cipher used in the satellite phones," *Sci. China Inf. Sci.*, vol. 61, no. 3, pp. 1–18, Mar. 2018, doi: 10.1007/s11432-017-9230-8.
[10] *GEO-Mobile Radio Interface Specifications; Part 1: General Specifications; Sub—Part 1: Abbreviations and Acronyms; GMR-2 01.004*, Standard TS 101 377-1-1, European Telecommunications Standards Institute (ETSI), 2001.
[11] *GEO-Mobile Radio Interface Specifications; Part 4: Radio Interface Protocol Specifications; Sub—Part 5: GMR-2 Mobile Earth Station—Network Interface; Data Link (DL) Layer Specifications; GMR-2 04.006*, Standard TS 101 377-4-5, European Telecommunications Standards Institute (ETSI), 2001.
[12] *GEO-Mobile Radio Interface Specifications; Part 4: Radio Interface Protocol Specifications; Sub—Part 7: Mobile Radio Interface Layer 3 Specifications; GMR-2 04.008*, Standard TS 101 377-4-7, European Telecommunications Standards Institute (ETSI), 2001.
[13] *GEO-Mobile Radio Interface Specifications; Part 5: Radio Interface Physical Layer Specifications; Sub—Part 3: Channel Coding; GMR-2 05.003*, Standard TS 101 377-5-3, European Telecommunications Standards Institute (ETSI), 2002.
[14] *Digital Cellular Telecommunication System (Phase 2+); General Description of a GSM Public Land Mobile Network (PLMN)*, Standard GSM 01.02, European Telecommunications Standards Institute (ETSI), 1996.
[15] J. L. Gross and J. Yellen, *Graph Theory and Its Applications*, 3rd ed. Boca Raton, FL, USA: CRC Press, 2018, doi: 10.1201/9780429425134.
[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
[17] *Data Encryption Standard (DES)*, Standard FIPS PUB 46-3, 1999.
[18] *GEO-Mobile Radio Interface Specifications; Part 4: Radio Interface Protocol Specifications; Sub—Part 7: Mobile Radio Interface Signalling Layer 3 General Aspects; GMR-1 04.007*, Standard TS 101 376-4-7, European Telecommunications Standards Institute (ETSI), 2002.

DONGJAE LEE is currently pursuing the Ph.D. degree with the Graduate School of Cyber Security, Korea University. His research interests include symmetric cryptography and post-quantum cryptography.

JAEWOO KIM is currently pursuing the B.S. degree in computer science and engineering with Seoul National University. His research interest includes software foundations.

DEUKJO HONG received the B.S. and M.S. degrees in mathematics and the Ph.D. degree in information security from Korea University, in 1999, 2002, and 2006, respectively. From 2007 to 2015, he was with the Electronics and Telecommunications Research Institute (ETRI). He is currently an Associate Professor with the Department of Information Technology and Engineering, Jeonbuk National University. His research interest includes symmetric cryptography.

JAECHUL SUNG received the Ph.D. degree in mathematics from Korea University, in 2002. He was a Senior Researcher with the Korea Information Security Agency (KISA), from July 2002 to January 2004. He is currently a Professor with the Department of Mathematics, University of Seoul. His research interests include cryptography, symmetric cryptosystems, hash functions, and MACs.

SEOKHIE HONG received the M.S. and Ph.D. degrees in mathematics from Korea University, in 1997 and 2001, respectively. He worked with Security Technologies Inc., from 2000 to 2004. Subsequently, he conducted Postdoctoral Research with COSIC, KU Leuven, Belgium, from 2004 to 2005, after which he joined the Graduate School of Cyber Security, Korea University. His research interests include cryptography, public and symmetric cryptosystems, hash functions, and MACs.