Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication*

Elad Barkan[1] Eli Biham[1] Nathan Keller[2]

[1] Computer Science Department
Technion – Israel Institute of Technology
Haifa 32000, Israel
Email: {barkan,biham}@cs.technion.ac.il
WWW: http://tx.technion.ac.il/∼barkan/,
http://www.cs.technion.ac.il/∼biham/

[2] Einstein Institute of Mathematics
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
Email: nkeller@math.huji.ac.il

Abstract. In this paper we present a very practical ciphertext-only cryptanalysis of GSM (Global System for Mobile communications) encrypted communication, and various active attacks on the GSM protocols. These attacks can even break into GSM networks that use "unbreakable" ciphers. We first describe a ciphertext-only attack on A5/2 that requires a few dozen milliseconds of encrypted off-the-air cellular conversation and finds the correct key in less than a second on a personal computer. We extend this attack to a (more complex) ciphertext-only attack on A5/1. We then describe new (active) attacks on the protocols of networks that use A5/1, A5/3, or even GPRS (General Packet Radio Service). These attacks exploit flaws in the GSM protocols, and they work whenever the mobile phone supports a weak cipher such as A5/2. We emphasize that these attacks are on the protocols, and are thus applicable whenever the cellular phone supports a weak cipher, for example, they are also applicable for attacking A5/3 networks using the cryptanalysis of A5/1. Unlike previous attacks on GSM that require unrealistic information, like long known-plaintext periods, our attacks are very practical and do not require any knowledge of the content of the conversation. Furthermore, we describe how to fortify the attacks to withstand reception errors. As a result, our attacks allow attackers to tap conversations and decrypt them either in real-time, or at any later time. We present several attack scenarios such as call hijacking, altering of data messages and call theft.
Keywords: GSM, cellular, ciphertext-only, cryptanalysis, GPRS, SMS, SIM, A5/2, A5/1.

1 Introduction

GSM is the most widely used cellular system in the world, with over a billion customers around the world. The system was developed during the late 1980s, and first GSM networks were deployed in the early 1990s. GSM is based on second generation cellular technology, i.e., it offers digitalized voice (rather than analog, as used in prior systems).

GSM was the first cellular system which seriously considered security threats. One example is a secure cryptographic hardware in the phone (the SIM - Subscriber Identity Module), which was introduced in GSM. Previous cellular systems had practically no security, and they were increasingly the subject of criminal activity such as eavesdropping on cellular calls, phone cloning, and call theft.

The security threat model of GSM was influenced by the political atmosphere around cryptology at the 1980s, which did not allow civilians to use strong cryptography. Therefore, the objective was

* An earlier version of this paper appears in CRYPTO 2003.

that the security of GSM would be equivalent to the security of fixed-line telephony. As a result, only the air-interface of GSM was protected, leaving the rest of the system un-protected. The aim of the protection on the air-interface was to provide two kinds of protections: protect the privacy of users (mostly through encryption), and protect the network from unauthorized access to the network (by cryptographic authentication of the SIM).

The privacy of users on the air-interface is protected by encryption. However, encryption can start only after the mobile phone identified itself to the network. GSM also protects the identity of the users by pre-allocating a temporary identification (TMSI - Temporary Mobile Subscriber Identity) to the mobile phone. This temporary identification is used to identify the mobile phone before encryption can commence. The temporary identification for the next call can safely be replaced once the call is encrypted.

Authentication of the SIM by the network occurs at a beginning of a radio conversation between the mobile phone and the network. After the phone identifies itself (e.g., by sending its TMSI), the network can initiate an authentication procedure. The procedure is basically a challenge-response scheme based on a pre-shared secret Ki between the mobile phone and the network. In the scheme, the network challenges the mobile phone with a 128-bit random number $RAND$; the mobile phone transfers $RAND$ to the SIM, which calculates the response $SRES = A3(Ki, RAND)$, where $A3$ is a one-way function; then, the mobile phone transmits $SRES$ to the network, which compares is to the $SRES$ value that it pre-calculated. The encryption key Kc for the conversation is created in parallel to the authentication by $K_c = A8(Ki, RAND)$, where $A8$ is also a one-way function (because $A3$ and $A8$ are invoked together, they are typically implemented by a single algorithm referred to as *A3A8*). The remainder of the call can be encrypted using K_c, and thus, the mobile phone and the network remain mutually "authenticated" due to the fact that they use the same encryption key. However, encryption is controlled by the network, and it is not mandatory. Therefore, an attacker can easily impersonate the network to the mobile phone using a false base station with no encryption. In general, it is not advisable to count on an encryption algorithm for authentication, especially in the kind of encryption that is used in GSM.

The exact design of A3 and A8 can be selected by each operator independently. However, many operators used the example, called *COMP128*, given in the GSM memorandum of understanding (MoU). Although never officially published, the design of COMP128 was reverse engineered by Briceno, Goldberg, and Wagner [8]. They have performed cryptanalysis of COMP128 [9], allowing to find the pre-shared secret Ki of the mobile phone and the network. Given Ki, A3 and A8 it is easy to perform cloning. Their attack requires the $SRES$ for about 2^{17} chosen values of $RAND$. The required data for this kind of attack can be obtained within a few hours over-the-air using a fake base station.

The original encryption algorithm for GSM was A5/1. However, A5/1 was export restricted, and as the network grew beyond Europe there was a need for an encryption algorithm without export restrictions. As a result, a new (weakened) encryption algorithm A5/2 was developed. The design of both algorithms was kept secret (it was disclosed only on a need-to-know basis, and under a non-disclosure agreement). In 2002, an additional new version A5/3 was added to the A5 family. Unlike A5/1 and A5/2, it's internal design was *published*. A5/3 is based on the block-cipher KASUMI, which is used in third generation networks [1].

The internal design of both A5/1 and A5/2 was reverse engineered from an actual GSM phone by Briceno [7] in 1999. The internal design was verified against known test-vectors, and it is available on the Internet [7].

After the reverse engineering of A5/1 and A5/2, it was demonstrated that A5/1 and A5/2 do not provide an adequate level of security for GSM. However, most of the attacks are in a known-plaintext attack model, i.e., they require the attacker not only to intercept the required data frames, but also to know their contents before they are encrypted.

A5/1 was initially cryptanalyzed by Golic [19] when only a rough outline of A5/1 was leaked. After A5/1 was reverse engineered, it was analyzed by Biryukov, Shamir, and Wagner [6]; Biham and Dunkelman [4]; Ekdahl and Johansson [11]; Maximov, Johansson and Babbage [20]; and recently by Barkan and Biham [2].

As for A5/2, it was cryptanalyzed by Goldberg, Wagner and Green [18] immediately after the reverse engineering. Their attack on A5/2 works in a negligible time complexity and it requires only two known-plaintext data frames which are exactly $26 \cdot 51 = 1326$ data frames apart (about 6 seconds apart). Another attack on A5/2 was proposed by Petrović and Fúster-Sabater [22]. This attack works by constructing a system of quadratic equations whose variables describe the internal state of A5/2 (i.e., equations of the form $c = \bigoplus_{i,j} a_i \cdot a_j$, where $a_i, a_j, c \in \{0, 1\}$, a_i and a_j are variables and c is a constant). This attack has the advantage that it requires only four known-plaintext data frames (thus the attacker is not forced to wait 6 seconds), but it does not recover the encryption key, rather, it allows to decrypt most of the remaining communications.

1.1 Executive Summary of the New Attacks

In this paper we describe several attacks on the A5 variants and on the GSM protocols. We first show a passive known-keystream attack on A5/2 that requires a few dozen milliseconds of known keystream. In this attack, we construct a system of quadratic equations that models the encryption process. Then, we solve the system to recover the internal state, and thus the key that was used.

We improve this attack on A5/2 to work in real time (finding the key in less than a second on a personal computer) by dividing the attack into two phases, a precomputation phase and a real-time phase. The attacker first performs a one-time precomputation of a few hours, in which he finds how to solve all the equation systems and stores instructions for the solution in memory. In the real-time phase, the attacker uses the instructions to quickly solve the equations.

Then, we transform this known-keystream attack on A5/2 into a ciphertext-only attack. The key idea is to take advantage of the fact that GSM employs error correction before encryption in the transmission path (instead of the well established reverse order). The error correction introduces linear dependencies between the bits. Assume that it is known that the parity (XOR) of some subset of bits is 0. XORing the same subset of bits after encryption reveals the parity of the corresponding keystream bits. We use an attack similar to the known-keystream attack, in which the parity of keystream bits is used instead of the keystream bits themselves. The resulting ciphertext-only attack completes in less than a second on a personal computer.

The above attacks assume that there are no reception errors. To overcome this restriction, we improve the attack on A5/2 to withstand a class of reception errors.

Next, we present a ciphertext-only attack on A5/1 whose complexity is considerably higher than the previous two attacks on A5/2. However, it demonstrates that passive A5/1 eavesdropping is feasible even for a medium-sized organization. We utilize the same technique as in the passive attack on A5/2, to reveal the parity of bits of the keystream. We then view the function from the internal state to the known-keystream bits as a random function, and apply it to a (generic) time/memory/data tradeoff attack, taken from the published literature [5]. Once the internal state is found, a candidate key is found (and can be checked using trial encryptions). It should be noted

that the time/memory/data tradeoff requires a lengthy preprocessing phase and a huge storage, but still the key can be recovered in a relatively short time. It should also be noted that the recovery process is probabilistic in nature, and that given enough data the success probability approaches one.

We then deal with another family of attacks, which are active attacks on the GSM protocol. These attacks can work even if the network supports only A5/1 or A5/3, as long as the mobile supports A5/2. The main flaw that facilitates the attacks is that the same encryption key is used regardless of whether the phone encrypts using A5/2, A5/1, or A5/3. Therefore, the attacker can mount a man-in-the-middle attack, in which the attacker impersonates the mobile to the network, and the network to the mobile (by using a fake base station). The attacker might use A5/1 for communication with the network and A5/2 for communications with the mobile, and due to the flaw, both algorithms encrypt using the same key, which the attacker can recover through a variant of the passive attack on A5/2. Since the attacker is in the middle, he can eavesdrop, change the conversation, perform call theft, etc. The attack applies to all the traffic including short message service (SMS).

A similar active attack applies to GPRS, which is a 2.5 generation service that allows mobile internet supporting services such as Internet browsing, e-mail on the move, and multimedia messages.

The security of GPRS is based on the same mechanisms as of GSM: the same A3A8 algorithm is used with the same Ki, but the authentication and key agreement of GPRS occurs in different times than in GSM, using another $RAND$ value called $GPRS\text{-}RAND$. Since the $RAND$ is different, the resulting $SRES$ and K_c are different, and are referred to as $GPRS\text{-}SRES$ and $GPRS\text{-}K_c$, respectively. The GPRS cipher is different from A5/1 and A5/2, and is referred to as GPRS-A5, or GPRS Encryption Algorithm (GEA). Similarly to A5, GEA is implemented in the phone (rather than in the SIM), thus an old SIM card can work in a GPRS-enabled phone. There are currently three versions of GEA: GEA1, GEA2, and GEA3 (which is similar to A5/3). Much like A5/1 and A5/2, the internal design of GEA1 and GEA2 was never made public.

An attacker can take advantage of the symmetry in the key agreement of GPRS and GSM by performing an active attack on the phone (using a fake base station). The attacker initiates a (non-GPRS) conversation with the mobile using A5/2, and send the $GPRS\text{-}RAND$ instead of $RAND$. Thus, the resulting key is identical to the key that is used in GPRS, and the attacker can recover it using the attack on A5/2.

1.2 Organization of this Paper

This paper is organized as follows: In Section 2, we give a short description of A5/2 and the way it is used. We present our new known plaintext attack in Section 3. This attack is improved in Section 4 to a ciphertext-only attack. We enhance our attack to withstand radio reception errors in Section 5. We then describe a passive ciphertext-only attack on A5/1 in Section 6. Active attacks on GSM are presented in Section 7, in which we show how to leverage the ciphertext-only attack on A5/2 to an active attack on any GSM network. We discuss the implications of the attacks under several attack scenarios in Section 8. Finally, we describe several ways of identifying and isolating a specific victim in Section 9. Section 10 summarizes the paper. In Appendix A, we improve Goldberg, Wagner, and Green's attack to a ciphertext-only attack. We give a technical background on GSM in Appendix B.

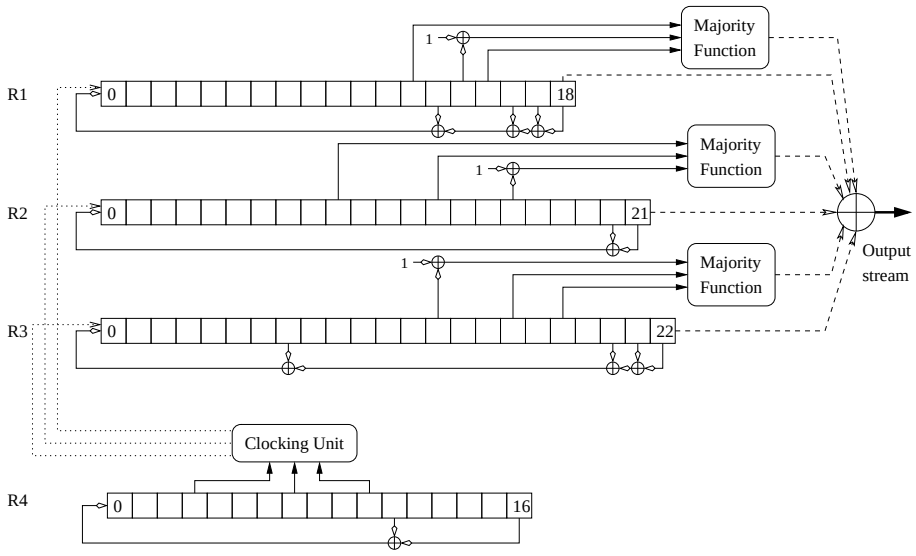

Fig. 1. The Internal Structure of A5/2

1. Set $R1 = R2 = R3 = R4 = 0$.
2. For $i = 0$ to 63
   - Clock all four registers.
   - $R1[0] \leftarrow R1[0] \oplus K_c[i]$; $R2[0] \leftarrow R2[0] \oplus K_c[i]$; $R3[0] \leftarrow R3[0] \oplus K_c[i]$; $R4[0] \leftarrow R4[0] \oplus K_c[i]$.
3. For $i = 0$ to 21
   - Clock all four registers.
   - $R1[0] \leftarrow R1[0] \oplus f[i]$; $R2[0] \leftarrow R2[0] \oplus f[i]$; $R3[0] \leftarrow R3[0] \oplus f[i]$; $R4[0] \leftarrow R4[0] \oplus f[i]$.
4. Set the bits $R1[15] \leftarrow 1$, $R2[16] \leftarrow 1$, $R3[18] \leftarrow 1$, $R4[10] \leftarrow 1$.

Fig. 2. The Key Setup of A5/2

2 Description of A5/2

The stream cipher A5/2 accepts a 64-bit key K_c, and a 22-bit publicly known initial value (IV) called COUNT (which is derived from the publicly known frame number, as described in Appendix B). We denote the value of COUNT by f. The internal state of A5/2 is composed of four maximal-length Linear Feedback Shift Registers (LFSRs): $R1$, $R2$, $R3$, and $R4$, of lengths 19-bit, 22-bit, 23-bit, and 17-bit, respectively, with linear feedback as shown in Figure 1. Before a register is clocked the feedback is calculated (as the XOR of the feedback taps). Then, the register is shifted one bit to the right (discarding the rightmost bit), and the feedback is stored into the leftmost location (location zero).

A5/2 is initialized with K_c and f in a four-step *key setup*, as described in Figure 2, where the i'th bit of K_c is denoted by $K_c[i]$, the i'th bit of f is denoted by $f[i]$, and $i = 0$ is the least significant bit. We denote the internal state after the key setup by $(R1, R2, R3, R4) = \text{keysetup}(K_c, f)$. Note

that the internal state after the key setup is linear in both the bits of K_c and f (without bits $R1[15]$, $R2[16]$, $R3[18]$, and $R4[10]$ that are always set to 1).

A5/2 works in cycles, where at the end of each cycle one output bit is produced. During each cycle two or three of registers $R1$, $R2$, and $R3$ are clocked by a clocking unit, based on the value of three bits of $R4$: $R4[3]$, $R4[7]$, and $R4[10]$. The clocking unit performs a majority function on the bits. Then, the registers are clocked as follows: $R1$ is clocked if and only if $R4[10]$ agrees with the majority. $R2$ is clocked if and only if $R4[3]$ agrees with the majority. $R3$ is clocked if and only if $R4[7]$ agrees with the majority. After these clockings, $R4$ is clocked, and an output bit is generated from the values of $R1$, $R2$, and $R3$, by XORing their rightmost bits to three majority values, one of each register. See Figure 1 for the exact details. It is important to note that the majority function (used for the output) is quadratic in its input as $maj(a, b, c) = a \cdot b \oplus b \cdot c \oplus c \cdot a$. Thus, an output bit is a quadratic function of bits of $R1$, $R2$, and $R3$.

The first 99 bits of output are discarded,[1] and the following 228 bits of output are used as the output keystream. The keystream generation can be summarized as follows:

1. Run the key setup with K_c and f (Figure 2).
2. Run A5/2 for 99 cycles and discard the output.
3. Run A5/2 for 228 cycles and use the output as keystream.

The output of 228 bits (referred to as keystream) is divided into two halves. The first half of 114 bits is used as a keystream to encrypt the link from the network to the phone, and the second half of 114 bits is used to encrypt the link from the phone to the network. Encryption is performed as a bitwise XOR of the message with the keystream.

It is worth noting that A5/2 is built on top of A5/1's architecture. The feedback functions of $R1$, $R2$ and $R3$ are the same as A5/1's feedback functions. The initialization process of A5/2 is also similar to that of A5/1 with just two differences: A5/2 also initializes $R4$, and one bit in each register is forced to be 1 after initialization. Then A5/2 discards 99 bits of output while A5/1 discards 100 bits of output. The clocking mechanism is the same, but the input bits to the clocking mechanism are from $R4$ in the case of A5/2, while in A5/1 they are from $R1$, $R2$, and $R3$. The designers meant to use similar building blocks to save hardware in the mobile phone [21].

3 Known Plaintext Attacks on A5/2

In this section we present a new known plaintext attack (known keystream attack) on A5/2. Namely, given a keystream divided into frames, and the respective frame numbers, the attack recovers the session key. For completeness we start by describing in detail Goldberg, Wagner, and Green's attack on A5/2.

3.1 Goldberg, Wagner, and Green's Known Plaintext Attack on A5/2

The first observation of this attack is that since $R4[10]$ is forced to be "1" during the key setup, $R4$ has the same value after key setup regardless of whether the bit $f[10]$ of COUNT is zero or one. Since $R4$ controls the clockings of $R1$, $R2$, and $R3$, the clockings of these registers are independent of the value of $f[10]$.

[1] Some references state that A5/2 discards 100 bits of output, and that the output is used with a one-bit delay. This is equivalent to stating that it discards 99 bits of output, and that the output is used without delay.

To mount an attack, the attacker aims to find two different frames with exactly the same clockings, i.e., two frames with f value which is identical up to $f[10]$. Due to the flaw in the key setup, the two frames would have exactly the same $R4$ value after the key setup. Taking into account the fixed permutation between the TDMA frame number and COUNT (the permutation is given in [17, annex C] as well as in Appendix B), two frames which are exactly $26 \cdot 51 = 1326$ TDMA frames (about 6 seconds) apart are required, where the first frame's $f[10]$ is zero. Note that if the first frame's $f[10]$ might is one, the attacker is forced to wait another six seconds for a frame with a zero $f[10]$. The attacker cannot use a frame with $f[10] = 1$ as a first frame, since due to the carry (remember that the TDMA frame number is incremented by one every frame) other bits of the COUNT are changed, and thus register $R4$ is different in the two frames. We conclude that the attacker is forced to wait between 6 to 12 seconds to obtain the required data for the attack.

The attack is as follows: Let f_1 and f_2 be the respective COUNT values for two frame numbers as described above, with respective key-streams k_1, k_2. Denote the values of registers $R1$, $R2$, $R3$, and $R4$ in the first frame, just after the key setup (before the 99 clockings), by $R1_1$, $R2_1$, $R3_1$, and $R4_1$, respectively. We use a similar notation for the initial internal state of the second frame, i.e., we denote the value of the registers in the second frame after the key setup by $R1_2$, $R2_2$, $R3_2$, and $R4_2$. Note that the special choice of f_1 and f_2 ensures that $R4_1 = R4_2$, and we denote its value by $R4$. The other registers are not equal, however, since the initialization process is linear in the bits of f_1 and f_2, the difference between $R1_1$, $R2_1$, $R3_1$ and $R1_2$, $R2_2$, $R3_2$, respectively, is also linear in the difference between f_1 and f_2. These differences are fixed, as $f_1 \oplus f_2 = 00000000000100000000000_b$. Thus, we can write $R1_1 = R1_2 \oplus \delta_1$, $R2_1 = R2_2 \oplus \delta_2$, $R3_1 = R3_2 \oplus \delta_3$, where δ_1, δ_2, and δ_3 are some constants.

We now show that given the value of $R4$, the keystream difference $k_1 \oplus k_2$ is linear in $R1_1$, $R2_1$, and $R3_1$. Given $R4$, the entire clocking of the registers is known (and is equal in the two frames as $R4_1 = R4_2$). Let l_1, l_2, and l_3 be the number of clocks that registers $R1$, $R2$, and $R3$ have been clocked by the end of cycle i. Therefore, the values of the three registers at the end of cycle i of the first frame are $L1^{l_1} \cdot R1_1$, $L2^{l_2} \cdot R2$, and $L3^{l_3} \cdot R3$, where $L1$, $L2$, and $L3$ are matrices that express one clocking of the respective registers. Similarly, the values of the registers at the second frame at the end of cycle i are $L1^{l_1} \cdot (R1_1 \oplus \delta_1)$, $L2^{l_2} \cdot (R2_1 \oplus \delta_2)$, and $L3^{l_3} \cdot (R3_1 \oplus \delta_3)$.

Let $g_1(R1) \oplus g_2(R2) \oplus g_3(R3)$ be the output bit of A5/2 given that the internal state of the registers is $R1$, $R2$, and $R3$; $g_1(\cdot)$, $g_2(\cdot)$, and $g_3(\cdot)$ are quadratic (as they involve one application of the majority function). To better understand that the output is quadratic in the internal state, consider the following example. Let $x_0, \ldots, x_{18}$, $y_0, \ldots, y_{21}$, $z_0, \ldots, z_{22}$ be variables representing the bits of R1, R2, and R3, respectively, just after the first bit of the keystream is produced. Then, the first bit of the keystream is

$$k_1[0] = x_{12}x_{14} \oplus x_{12} \oplus x_{12}x_{15} \oplus x_{14}x_{15} \oplus x_{15} \oplus x_{18} \oplus y_9y_{13} \oplus \ldots \oplus z_{16}z_{18} \oplus z_{22}$$

(which is quadratic in the variables representing the internal state).

Goldberg, Wagner, and Green observed that the difference of the output bits can be expressed as a linear function of the internal state of the first frame. The difference in the output bit of cycle i is given by:

$$\begin{aligned} &g_1(L1^{l_1} \cdot R1_1) \oplus g_1(L1^{l_1} \cdot R1_1 \oplus \delta_1) \oplus \\ &g_2(L2^{l_2} \cdot R2_1) \oplus g_2(L2^{l_2} \cdot R1_2 \oplus \delta_2) \oplus \\ &g_3(L3^{l_3} \cdot R3_1) \oplus g_3(L3^{l_3} \cdot R1_3 \oplus \delta_3) = g_{\delta_1}(L1^{l_1} \cdot R1_1) \oplus g_{\delta_2}(L2^{l_2} \cdot R2_1) \oplus g_{\delta_3}(L3^{l_3} \cdot R3_1), \end{aligned}$$

where $g_{\delta_1}(\cdot)$, $g_{\delta_2}(\cdot)$, and $g_{\delta_3}(\cdot)$ are defined in the sequel and shown to be linear functions. Thus, the output difference is linear in $R1_1$, $R2_1$, and $R3_1$. It remains to show that given a quadratic function $g(x_1, \ldots, x_n)$ and $\Delta = \Delta_1, \ldots, \Delta_n$, where $x_i, \Delta_i \in \{0, 1\}$, the function $g_\Delta = g(x_1, \ldots, x_n) \oplus g(x_1 \oplus \Delta_1, x_2 \oplus \Delta_2, \ldots, x_n \oplus \Delta_n)$ is linear in $x_1, \ldots x_n$. Note that in the above equations $\Delta \in \{\delta_1, \delta_2, \delta_3\}$.

Since g is quadratic, it can be written as

$$g(x_1, \ldots, x_n) = \sum_{1 \le i,j \le n} a_{i,j} x_i x_j \oplus a_{0,0},$$

where $a_{i,j} \in \{0, 1\}$ are fixed for a given g and recalling that $x_i x_i = x_i$. Thus,

$$\begin{aligned} g_\Delta &= \sum_{1 \le i,j \le n} a_{i,j}(x_i x_j \oplus (x_i \oplus \Delta_i)(x_j \oplus \Delta_j)) \\ &= \sum_{1 \le i,j \le n} a_{i,j}(x_i x_j \oplus x_i x_j \oplus x_i \Delta_j \oplus \Delta_i x_j \oplus \Delta_i \Delta_j) \\ &= \sum_{1 \le i,j \le n} a_{i,j}(x_i \Delta_j \oplus \Delta_i x_j \oplus \Delta_i \Delta_j). \end{aligned}$$

The last expression is linear in $x_1, \ldots, x_n$ given $\Delta_1, \ldots, \Delta_n$.

Therefore, given $R4$ and $k_1 \oplus k_2$, the initial internal state $R1_1$, $R2_1$, and $R3_1$ can be recovered (by solving a linear system of equations). K_c can be recovered from the initial internal state ($R1_1$, $R2_1$, $R3_1$, $R4_1$) and f_1 by reversing the key setup of A5/2. As $R4$ is not known, the attacker needs to guess all possible 2^{16} values of $R4$ (recall that $R4[10]$ is set to one, and thus does not need to be guessed), and for each value solve the resulting linear equation, until a consistent solution is found.

A faster solution is possible by filtering for the correct $R4$ values. The initial internal state of $R1$, $R2$, and $R3$ is 61 bits (recall that three bits of $R1$, $R2$, and $R3$ are set to 1). Thus, 61 bits of $k_1 \oplus k_2$ are required to reconstruct K_c, while $k_1 \oplus k_2$ is 114 bits long. It is therefore possible to construct an overdetermined linear system whose solution is the internal state. The $114 - 61 = 53$ dependent equations would zero during the Gauss elimination. These equations depend on the value of $R4$, thus, for each value of $R4$, it is possible to write 53 equations $V_{R4} \cdot (k_1 \oplus k_2) = 0$, where V_{R4} is a 53×114 bits matrix, and 0 is a vector of 53 zeros. The redundancy is used to filter wrong $R4$ values for which $V_{R4} \cdot (k_1 \oplus k_2) \neq 0$. On average it takes two dot products (out of the 53 equations) to disqualify a wrong $R4$ value. As there are 2^{16} possible values for $R4$, and as on average the correct $R4$ would be found after trying $2^{16}/2$ values, the average attack time is about 2^{16} dot products, plus a single solution of the equation system. A straightforward implementation on a 32-bit personal computer, where all possible V_{R4} systems are pre-loaded to memory, consumes $2^{16}(16 \cdot 114)/8 = 2^{16} \cdot 228$ bytes (about 15 MBs) of volatile memory, and requires a few milliseconds of CPU time (on a 2GHz personal computer) to filter for the correct value of $R4$. Once $R4$ is found, we can solve the linear equations for this specific $R4$ in order to recover $R1_1$, $R2_1$, and $R3_1$. Storing these systems of equations after Gauss elimination takes about $2^{16} \cdot 64 \cdot 114/8 = 2^{16} \cdot 912$ bytes, i.e., about 60 MBs of memory. Note that this memory can be stored on a hard-disk, and can be indexed by $R4$. Given $R4$, the relevant system can be fetched to volatile memory. The complexity can be further reduced by considering fewer bits of $k_1 \oplus k_2$.

The attack as described above requires a relatively short preprocessing consisting of the computation of the equations and the V_{R4} matrices. The preprocessing can be completed within a few minutes on a personal computer.

3.2 Our Non-Optimized Known-Plaintext Attack on A5/2

We present an attack on A5/2 that requires the keystream of (any) four frames. Our attack recovers the internal state ($R1$, $R2$, $R3$, and $R4$), and by reversing the key setup, it finds the session key.

Our known-plaintext attack can be viewed as an improvement of Goldberg, Wagner, and Green's attack. We guess the initial value of $R4$, and write every output bit as a quadratic term in $R1$, $R2$, and $R3$. We describe a way to write every output bit — even if on different frames — as a quadratic term of $R1$, $R2$, and $R3$ of the first frame. Given the output bits of four frames, we construct a system of quadratic equations, and solve it using linearization. Thus, we recover the initial value of $R1$, $R2$, and $R3$.

Let k_1, k_2, k_3, and k_4 be the keystream of A5/2 for frames f_1, f_2, f_3, and f_4, respectively. Note that each k_j is the output keystream for a whole frame, i.e., each k_j is 114-bit long.[2] We denote the i'th bit of the keystream of f_j by $k_j[i]$. The initial internal state of register Ri of frame f_j (after the initialization but before the 99 clockings) is denoted by Ri_j.

As we discussed in Section 3.1, given $R4$, each output bit can be written as a quadratic function of the initial internal state of $R1$, $R2$, and $R3$. We would like to construct a system of quadratic equations that expresses the equality of the quadratic terms for each bit of the output, and the actual value of that bit from the known-keystream. The solution of such a system would reveal the initial internal state. However, solving a general system of quadratic equations is NP complete. Fortunately, there are shortcuts when the quadratic system is overdefined (in our case there are 61 variables and 114 quadratic equations, so the system is overdefined). The complexity drops significantly as the system becomes more and more overdefined. Therefore, we improve this attack by adding equations from other frames, while making sure the equations are over the same variables, i.e., the initial value of $R1$, $R2$, $R3$ at frame f_1. Once we combine the equations of four frames, we solve the system by linearization.

A system of equations is built for each of the 2^{16} possible values for $R4_1$ and solved, until we find a consistent solution. The solution of such a system is the initial internal state at frame f_1.

There are at most 656 variables after linearization: We observe that each majority function operates on bits of a single register. Therefore, the quadratic terms consist of pairs of variables of the same register only. Taking into account that one bit in each register is set to 1, $R1$ contributes 18 linear variables and all their $\frac{17 \cdot 18}{2} = 153$ products (the linearization implies that each product is treated as a new variable). In the same way $R2$ contributes $21 + \frac{21 \cdot 20}{2} = 21 + 210$ variables and $R3$ contributes $22 + \frac{22 \cdot 21}{2} = 22 + 231$ variables, totaling $18 + 153 + 21 + 210 + 22 + 231 = 655$ variables after linearization. We include the constant 1 as a variable to represent the affine part of the equations, thus our set of variables contains 656 variables. We denote the set of these 656 variables for frame f_i by S_i.

It remains to show how given the variables in the set S_1 of frame f_1, we can describe the output bits of frames f_2, f_3, and f_4 as linear combinations of variables from the set S_1. Assume that we know the value of $R4_1$, and recall that the key setup is linear in COUNT (see Section 2) (and that COUNT is publicly known for both frames). Therefore, given the COUNT difference of the frames, we know the difference in the values of each register after key setup: $R4_1$ is given, and thus we know $R4_2$. As $R1_1$, $R2_1$, and $R3_1$ are unknown, we only know the XOR-differences between $R1_1, R2_1, R3_1$ and $R1_2, R2_2, R3_2$ respectively.

[2] Note that by keystream for a frame, we refer to the 114-bit keystream half that is used in the encryption process of the frame for a single direction, e.g., the network-to-mobile link.

We translate each variable in S_2 to variables in S_1: Let α_1 be the concatenated value of the linear variables in S_1, and g a quadratic function such that $S_1 = g(\alpha_1)$. We know that the concatenated value of the linear variables of S_2 can be written as $\alpha_2 = \alpha_1 \oplus \delta_{1,2}$, and clearly $S_2 = g(\alpha_2)$. Much like in Section 3.1, the difference between S_2 and S_1 is linear in the bits of α_1, which implies that S_2 can be expressed in linear terms of the variables in S_1. Thus, we construct a system of quadratic equations using the keystream of four frames with the variables taken only from S_1. In total, we create an equation system of the form: $S_{R4_1} \cdot S_1 = k$, where S_{R4_1} is the system's matrix of dimension 456×656, $k = k_1||k_2||k_3||k_4$ (it is 456 bits long), and "||" denotes concatenation. Note that S_{R4_1} depends on the value of $R4_1$, and on the difference between the COUNT value of the frames.

Clearly, once we obtain 656 linearly independent equations the system can be easily solved using Gauss elimination. We observe that it is practically very difficult to collect 656 *linearly independent* equations, due to the low order of the output function and the frequent initializations of A5/2 (A5/2 is re-initialized once 228 output bits are generated). However, we do not actually need to solve all the variables, as it suffices to solve the linear variables of the system. We have tested experimentally and found that about 450 linearly-independent equations are always sufficient to solve the original linear variables in S_1 using linearization and Gauss elimination.[3]

It is interesting to see that we can gain 13 additional linear equations for free, due to the knowledge of $R4_1$, and the frame number. Let $R1234_1 \triangleq R1_1||R2_1||R3_1||R4_1$, where '||' denotes concatenation. We treat $R1234_1$ as a 77-bit vector, throwing away the four bits that are set to 1 during the key setup. $R1234_1$ is linear in the bits of K_c and f_1, i.e., we can write

$$R1234_1 = N_K \cdot K_c \oplus N_f \cdot f_1, \tag{1}$$

where N_K is a 77×64 matrix, and N_f is a 77×22 matrix that represents the key setup. The linear space which is spanned by the columns of N_k is of dimention 64, but each vector in that space has 77 bits, therefore, 13 linear equations always hold on $N_K \cdot K_c$; let H_K be the matrix 13×77 that expresses these equations, i.e.,

$$H_K \cdot N_K = 0,$$

where 0 is the 13×64 zero matrix. We multiply Equation (1) on the left by H_K:

$$H_K \cdot R1234_1 = H_K \cdot N_K \cdot K_c \oplus H_K \cdot N_f \cdot f_1 = H_K \cdot N_f \cdot f_1.$$

We can divide H_K into two parts H_K^L and H_K^R such that

$$H_K \cdot R1234_1 = H_K^L \cdot R123_1 \oplus H_K^R \cdot R4_1,$$

where $H_K = H_K^L||H_K^R$, H_K^L is 13×61 (the leftmost 61 columns of H_K), H_K^R is 13×16 (the rightmost 16 columns of H_K), and $R123_1 = R1_1||R2_1||R3_1$. It follows that

$$H_K \cdot N_f \cdot f_1 = H_K \cdot R1234_1 = H_K^L \cdot R123_1 \oplus H_K^R \cdot R4_1,$$

which we can reorganize to:

$$H_K^L \cdot R123_1 = H_K \cdot N_f \cdot f_1 \oplus H_K^R \cdot R4_1.$$

[3] In case the data available for the attacker is scarce, there are additional methods that can be used to reduce the number of required equations. For example, whenever a value of a linear variable x_i is discovered, any quadratic variable of the form $x_i \cdot x_j$ can be simplified to 0 or x_j depending whether $x_i = 0$ or $x_i = 1$, respectively. The XL algorithm [10] can also be used in cases of scarce data.

Namely, given $R4_1$ and the relevant COUNT (i.e., f_1), we gain 13 linear equations (H_K^L) over the bits of registers $R1$, $R2$, and $R3$, i.e., over $R123_1$.

We summarize the attack of this section as follows: we try all the 2^{16} possible values for $R4_1$, and for each such value, we solve the linearized system of equations that describe the output bits for four frames. The solution of each system gives us a suggestion for the internal state of R1, R2, and R3, which together with R4 is a suggestion for the full internal state. Most of the $2^{16} - 1$ wrong states can be easily identified due to inconsistencies in the Gauss elimination. If two or more consistent internal states remain, they are verified by trial encryptions.

The time complexity of the attack is as follows: There are 2^{16} guesses of the value of $R4_1$. For each guess, we solve a linear binary system of 656 variables, which is about $656^3 \approx 2^{28}$ XOR operations. Thus, the total complexity is about 2^{44} bit-XOR operations. When performed on a 32-bit machine, the complexity is 2^{39} register-XOR operations.

An implementation of this algorithm on a Linux 800MHz Pentium III personal computer finds the internal state within about 40 minutes, and requires relatively small amount of memory (holding the linearized system in memory requires 656^2 bits $\approx 54KB$).

3.3 An Optimized Attack on A5/2

We now describe an optimized implementation of the attack. The optimized version of the attack finds K_c in a few milliseconds of CPU time, and uses precomputed tables stored in memory. However, it requires slightly more data compared to the un-optimized attack.

The key idea of the optimized attack is similar to the one used in Section 3.1 for a faster attack: In a precomputation phase, we compute the dependencies that occur during the Gauss elimination of the system of equations for each $R4_1$ value. Then, in a real-time phase, we filter for the correct $R4_1$ value by applying the consistency checks on the known keystream, and keeping only the $R4_1$ values that are consistent with the keystream.

In other words, we perform a precomputation phase, in which we calculate the equation systems for all values of $R4_1$ in advance. We solve each such system in advance, i.e., given a system of equations $S_{R4_1} \cdot S_1 = k$, we compute a "solving matrix" T_{R4_1}, such that $T_{R4_1} \cdot S_{R4_1}$ is the result of Gauss elimination of S_{R4_1}. Since S_{R4_1} does not only depend on $R4_1$ but also on the difference between the COUNT values of the frames, we have to perform the precomputation for several COUNT value differences, as we discuss later. In the real-time phase, we calculate $t = T_{R4_1} \cdot k$ for each value of $R4_1$. The first elements of the vector t are the (partially solved) variables in S_1, but as some of the equations are linearly dependent (described in Section 3.2), the remaining elements of t should be zeros (representing the dependent equations). Therefore, we check that the last elements in t are indeed zero, i.e,. that the keystream k is consistent with the tested value for $R4_1$. Once a consistent value for $R4_1$ is found, we can verify it by calculating the key and performing trial encryptions. In an even faster implementation, we do not need to hold in memory the entire matrices T_{R4_1}. We only hold the last rows $T^0_{R4_1}$ of the matrices T_{R4_1}, i.e., the rows that correspond to the zero elements in t. Then, to verify consistency of a value $R4_1$, we only need to check that $t' = T^0_{R4_1} \cdot k$ is a vector of zeros. We do not need to keep more than 16 rows in $T^0_{R4_1}$, as 16 would ensure that on the average case there would be two values of $R4_1$ that are consistent, one of them is the correct $R4_1$.

We now analyze the time and memory complexity of the attack using a single precomputed table (for a single difference between the COUNT value of the frames). The time that is required for the precomputation is comparable to performing the un-optimized attack, i.e., takes about 40 minutes

on our computer. In the real-time phase, we must keep the filtering matrices in volatile memory for fast operation. A single system matrix is about $456 \cdot 16$ bits, thus, about 60 MBs are required to hold the table for the 2^{16} possible values of $R4_1$. Additional $64 \cdot 456 \cdot 2^{16} \approx 240$ MBs are required to hold the matrices that are used to find the full internal state given $R4_1$ and the keystream. However, these matrices can be stored on a hard drive. The attack time is about 250 CPU cycles for multiplying and checking a single matrix, or about 16M cycles in total (a few milliseconds on a personal computer). The limiting factor is the bus speed between the memory and the CPU. After finding an $R4_1$ candidate, loading the relevant solution matrix from disk takes another few tens of milliseconds (and a negligible time to find K_c). In our implementation, the attack takes less than a second on a personal computer.

As we mentioned, S_{R4_1} depends on the value of $R4_1$ and on the difference between the COUNT value of the different frames, i.e., when we perform the precomputation, we must know the XOR difference between the COUNT values of the frames. The difference between the COUNT values is used while translating the sets of variables S_2, S_3, and S_4, to S_1.

We satisfy the requirement of knowing in advance the XOR difference between the COUNT values of the frames as follows: We perform the precomputation several times, for different possible differences, and store the results in different tables. Then, in the real-time phase, we use the tables that are appropriate for the COUNT values of our frames. If we are given known keystream for frames with COUNT values that is not covered by our precomputation, then we are forced to abandon this keystream, and wait a for keystream with COUNT difference as we precomputed.

From this point to the end of the section, we give a technical example of a real GSM channel and how we deal with the requirement of knowing in advance the XOR difference between COUNT values. Consider the downlink of the SDCCH/8 channel (see Appendix B for more details about the channel). This channel is used many times in GSM call initiation, even before the mobile phone rings. In this channel, a message is transmitted over four consecutive frames out of a cycle of 51 frames. The four frames are always transmitted on the same values of the frame number modulo 51 and starting when the two least significant bits of the frame number modulo 51 are zero. Clearly, the frame number modulo 26 can take any value between zero to 25 (and it is actually decreased by one every cycle as $51 \equiv -1 \pmod{26}$). Let fr denote the first frame number of these four frames, i.e., the four frames are $fr_1 = fr$, (and the two lower bits of $fr \pmod{51}$ are zero) $fr_2 = fr + 1$, $fr_3 = fr + 2$, and $fr_4 = fr + 3$. Detailed analysis shows that by repeating the precomputation for specific 13 values of fr mod 26, a success rate of 100% is reached. Alternatively, we can perform the precomputation for only some of the values, and discard some frames until the received frames match the ones meeting the pre-computed conditions.

During the precomputation for a specific fr in the downlink SDCCH/8, the differences $fr \oplus fr_2 \pmod{26}$, $fr \oplus fr_3 \pmod{26}$, and $fr \oplus fr_4 \pmod{26}$ must be fixed. By performing precomputation for the cases where the lower bits of fr mod 26 are $00, 001, 010$, and 011 we cover the XOR-difference for the cases where the first frame number fr modulo 26 is $0, 1, 2, 3, 4, 8, 9, 10, 11, 12, 16, 17$, 18, 19, and 20. When the lower bits $fr \pmod{26}$ are 0101, we cover the cases where fr mod 26 is: 5 and 21. When the lower bits fr mod 26 are 0110, we cover fr mod 26 values 6 and 22. We cover each of the following fr mod 26 values by its own: 7, 13, 14, 15, 23, 24, 25. Thus, by repeating the precomputation 13 times we build a full coverage, i.e., given the output of A5/2 for four consecutive frames, we use the relevant precomputed tables to perform the attack. Alternatively, we can perform precomputation only for some of the possible values of fr mod 26, and during the attack, discard frames until we reach a set of four frames whose differences are covered by the precomputation. For

example, if we precompute the equation systems for the cases where the lower bits of fr mod 26 are 00, then the following fr mod 26 values are covered by the tables: $0, 4, 8, 12, 16, 20$. The worst case is when fr mod 26 equals 25. In this case, the next quartets of frames begin with fr mod 26 of $24, 23, 22, 21$, i.e., we throw five quartets of frames, and perform the attack using the sixth quartet for which fr mod 26 equals 20 (i.e., we waste about 1.1 second of data).

In the above example of the SDCCH/8, a full optimized implementation requires the keystream of four consecutive frames. After a one-time precomputation of about $40 \cdot 13 = 520$ minutes, and using 780 MBs of RAM, and another 3.1 GBs on disk, the attack works in less than a second. Note that we can refrain from saving the K_c matrices, and thus save 3.1 GBs on the hard-disk, and in return recompute the system of equations for the correct $R4_1$, once found (in this case the total attack time is still less than one second on a personal computer).

4 An Instant Ciphertext-Only Attack on A5/2

In this section, we transform the attacks of Section 3.2 and Section 3.3 to a ciphertext-only attack on A5/2.

GSM must use error correction to withstand reception errors. However, during the transmission, a message is first subjected to an error-correction code, which considerably increases the size of the message. Only then, the coded message is encrypted and transmitted (see [16, Annex A]). This transmission path contradicts the common practice of first encrypting a message, and only then subjecting it to error-correction codes. Some readers may wonder how it is even possible to correct errors (on the reception path) after decryption, as decryption often causes single bit errors to propagate through the entire message. However, since GSM decrypts by bitwise XORing the keystream to the ciphertext, an error in a bit before decryption causes an error in the corresponding bit after decryption, without any error-propagation. This trick of reversing the order of encryption and error-correction would not have been possible if a block-cipher was used for encryption. Subjecting a message to error-correction codes before encryption introduces a structured redundancy in the message, which we use to mount a ciphertext-only attack.

There are several kinds of error-correction methods that are used in GSM, and different error-correction schemes are used for different channels (see [12] for exact description of GSM channel coding). For readers unfamiliar with GSM channels, we recommend reading Appendix B. However, most of this section is intelligible without reading the appendix.

We focus on the error-correction codes of the Slow Associated Control Channel (SACCH), which is also used in the $SDCCH/8$ channel. Both channels are commonly used in the beginning of the call. Other channels are used in other stages of the conversation, and our attack can be adapted to these channels (although it's enough to find the key on the $SDCCH/8$ at the beginning of the call, as the key does not change during the course of a conversation).

In the SACCH, the message to be coded with error-correction codes has a fixed size of 184 bits. After the error-correction codes are employed, the result is a 456-bit long message. The 456 bits of the message are then interleaved, and divided into four frames. These frames are then encrypted and transmitted.

The coding operation and the interleaving operation can be modeled together as a multiplication of the message (represented as a 184-bit binary vector, and denoted by P) by a constant 456×184 matrix over $GF(2)$, which we denote by G, and XORing the result with a constant vector denoted by g. The result of the coding-interleaving operation is: $M = (G \cdot P) \oplus g$. The vector M is divided into

four data frames. In the encryption process, each data frame is XORed with the output keystream of A5/2 for the respective frame.

Since G is a 456×184 binary matrix, there are $456 - 184 = 272$ equations that describe the kernel of the inverse transformation. The dimension of the kernel is exactly 272 due to the properties of the matrix G. In other words, for any vector $M \oplus g$, such that $M = G \cdot P \oplus g$, there are 272 linearly independent equations on its elements. Let H be a matrix that describes these 272 linear equations, i.e., $H \cdot (M \oplus g) = 0$ for any such M (In coding theory such H is called the parity-check matrix).

We now show how to use the redundancy in M to mount a ciphertext-only attack. The key observation is that given the ciphertext, we can evaluate the linear equations over the underlying keystream bits. Recall that the ciphertext C is computed by $C = M \oplus k$, where $k = k_1||k_2||k_3||k_4$ is the keystream of the four frames, and "||" denotes concatenation. We use the same 272 equations on $C \oplus g$, namely:

$$H \cdot (C \oplus g) = H \cdot (M \oplus k \oplus g) = H \cdot (M \oplus g) \oplus H \cdot k = 0 \oplus H \cdot k = H \cdot k.$$

Since the ciphertext C is known (and g is fixed and known), we actually have linear equations over the bits of k. Note that the linear equations are independent of P — they depend only on k. Thus, we now have a linear equation system over the bits of the keystream. For each guess of $R4_1$, we substitute each bit of k in this equation system with its description as linear terms over S_1 (see Section 3.2), and thus get a system of equations on the 656 variables of S_1. Each 456-bit coding block provides 272 equations, hence after two blocks, we have more than 450 equations. In a similar way to the attack of Section 3.2, we perform Gauss elimination, and about 450 equations are enough to find the value of all the original linear variables in S_1. K_c is then found by inverting the key setup of A5/2.

The rest of the details of the attack and its time complexity are similar to the case in the previous sections. The major difference is that in the known-plaintext attacks we know the keystream bits, and in the ciphertext-only attack, we know only the value of linear combinations of keystream bits (through the ciphertext and error-correction codes). Therefore, the resulting equations in the ciphertext-only attack are linear combinations of the equations in the known-plaintext attack: Let $S_{R4_1} \cdot S_1 = k$ be a system of equations from Section 3.3, where S_{R4_1} is the system's matrix. In the ciphertext-only attack, we multiply this system by H on the left as follows: $(H \cdot S_{R4_1}) \cdot S_1 = (H \cdot k)$. Recall that H is a fixed known matrix that depends only on the coding-interleaving matrix G, and that $H \cdot k$ is computed from the ciphertext as previously explained. Therefore, we can solve this system and continue like in previous sections. In the known-keystream attack, we try all the 2^{16} possible equation systems S. In the ciphertext-only attack, we try all the 2^{16} possible equation systems $H \cdot S_{R4_1}$ instead. In the pre-computation of the optimized ciphertext-only attack, for such system we find linear dependencies of rows by a Gauss elimination. In the real-time phase of the ciphertext-only attack, we filter wrong values of $R4_1$ by checking if the linear dependencies that we found in the pre-computation step hold on the bits of $H \cdot k$.

A technical difference between the ciphertext-only attack and the known plaintext attacks is that while four frames of known plaintext provide enough equations, about eight ciphertext frames are required in the ciphertext-only attack. The reason is that in the ciphertext-only attack, from 456 bits of ciphertext, we extract only 272 equations. A consequence of using eight frames instead of four in the optimized version of the attack is that the constraint on the XOR differences of the frame numbers is stronger, as we need to know in advance the XOR differences between eight frames

(instead of four in the case of known-keystream). This constraint has a very slight implications, for example, in the case of the SDCCH/8 channel, it increases the number of precomputations that need to be performed to 16 (compared to 13 in the optimized known-plaintext attack). However, depending on the attack configuration, with a small probability we might need extra four frames of data (as T1 might change, see Appendix B).

We summarize that the time complexity of an optimized ciphertext-only attack is identical to the case of the optimized known-plaintext attack. The preprocessing and memory consumption of the optimized attack (in case of the downlink SDCCH/8 channel) is $16/13 \approx 1.23$ times the respective complexity of the known plaintext attack. We have implemented a simulation of the attack, and verified these results.

Our methods allow to enhance the attack of Goldberg, Wagner, and Green and the attack of Petrović and Fúster-Sabater to ciphertext-only attacks. We give a description of the enhancement of Goldberg, Wagner, and Green's attack in Appendix A.

5 Withstanding Errors in the Reception

A possible problem in a real-life implementation of the attacks is the existence of radio reception errors. A single flipped bit might fail an attack (i.e., the attack ends without finding K_c). Once the attack fails, the attacker can abandon the problematic data, and start again from scratch. But in a noisy environment, the chances are high that the new data will also contain errors. An alternative approach that we present in this section is to correct these errors.

Two kinds of reception errors can occur: flipped bits, and erasures. A flipped bit is a bit that was transmitted as "1" and received as "0", or vice versa. Erasures occur when the receiver cannot determine whether a bit is "1" or "0". Many receivers can report erased bits (rather than guessing a random value).

A possible inefficient algorithm to correct reception errors exhaustively tries all the possibilities for errors. For flipped bits, we can first try to employ the attack without any changes (assuming no errors occur), and if the attack fails, we repeat it many times, each time we guess different locations for the flipped bits. We try the possibilities with the least amount of errors first. The time complexity is exponential in the number of errors, i.e., about $\binom{n}{e}A$, where A is the time complexity of the original attack, n is the number of input bits, and assuming we know there are e flipped bits.

Correcting erasures is somewhat easier, as we only need to try all the possible values for the erased bits. The time complexity is thus $2^e A$, where e is the number of erasures. An even easier solution exists in the un-optimized known-plaintext attack, in which an erased plaintext bit translates to an erased keystream bit. Each keystream bit contributes one equation, thus, we can simply remove the equations of the erased keystream bits. If not too many erasures occur, we still have sufficiently many equations to perform the attack. However, in the optimized attack, we precompute all the equation systems, and thus we cannot remove an equation a posteriori. We could pre-compute the equation systems for every possible erasure pattern, but it would take a huge amount of time to compute and a huge amount of storage. Therefore, another method is needed.

In the rest of this section, we present an (asymptotically) better method to apply the optimized attack with the presence of erasures. For simplicity, we focus on the optimized known-plaintext attack on A5/2, but note that the optimized ciphertext-only attack can be similarly improved.

Assume that e erasures occur with their locations known, but our data is free of bit flips. We view the keystream as the XOR of two vectors, the first vector contains the undoubted bits of

the keystream (with the erased bits set to zero), and the second vector has a candidate value for the erased bits (with the undoubted bits set to zero). Let r be the first vector. Let w_i be the i^{th} candidate (out of the 2^e possibilities) for the second vector, where i is the binary value of the concatenated erased bits. Thus, given the correct value for i, the correct keystream is $k = r \oplus w_i$.

The correct value of i can be found without an exhaustive search. Recall the consistency-check matrices T_{R4_1} of Section 3.3. The linear space spanned by $T_{R4_1} \cdot w_i$, where $i \in [0, \ldots, 2^e - 1]$, has a maximum dimension of e (if the columns of T_{R4_1} are linearly independent the dimension is exactly e, for simplicity we assume that this is indeed the case). We denote this linear space by $\boldsymbol{T_{R4_1}}$.

We reduce the problem of finding the correct i to a problem of solving a linear system. For each candidate $R4_1$, we compute $T_{R4_1} \cdot r$. Clearly, for the correct $R4_1$ value and for the correct w_i value, $T_{R4_1} \cdot (w_i \oplus r)$ is a vector of zeros. Therefore, for the correct w_i, $T_{R4_1} \cdot w_i = T_{R4_1} \cdot r$. Thus, the problem of finding the correct i is reduced to finding the w_i that solves this equation.

An efficient way to solve such a system is as follows: First find e vectors that span the space $\boldsymbol{T_{R4_1}}$. Such e vectors are given by $b_j = T_{R4_1} \cdot w_{2^j}$, where $j \in \{0, 1, 2, \ldots, e-1\}$. Then, we define a new matrix B whose columns are the vectors b_j: $B = (b_0, \ldots, b_{e-1})$. Finally, we find the correct i by requiring that $B \cdot i = T_{R4_1} \cdot r$, and solving the system (e.g., using Gauss elimination) to find i. If inconsistencies occur during the Gauss elimination, we move on to the next candidate $R4_1$, otherwise we assume we found the value of $R4_1$ and the keystream, and use the attack to recover K_c (which is verified using a trial encryption). Note that if the dimension of $\boldsymbol{T_{R4_1}}$ is smaller than e, then Gauss elimination might result in more than one option for i. In such case, the number of options for i is always less or equal to 2^e.

The number of needed rows in T_{R4_1} in order to correct e erasures is about $16 + e$: For each of the 2^{16} candidate values of $R4_1$ the e erasures span a space of at most 2^e vectors, thus, there are about 2^{16+e} candidate solutions. Therefore, the number of rows in T_{R4_1} needs to be about $16 + e$ in order to ensure that only about two consistent solutions remain.

The time complexity of correcting the erasures for a single candidate of $R4_1$ is composed of first calculating the matrix B and $T_{R4_1} \cdot r$, and then solving the equation system $B \cdot i = T_{R4_1} \cdot r$. Calculating B and $T_{R4_1} \cdot r$ is comparable to one full vector by matrix multiplication, i.e., about $456(16 + e)$ bit-XORs. The Gauss elimination takes about $O((16 + e)^3)$ bit-XOR operations. The processes is repeated for every possible value of $R4_1$. Thus, the time complexity is about $2^{16}(456(16+e)+(16+e)^3)$ bit-XOR operations. Assuming that ten erasures need to be corrected, the total time complexity is about 2^{31} bit-XOR operations, i.e., about three and a half times the complexity of the optimized known-plaintext attack without reception errors. A naive implementation for correcting ten erasures would take about $2^{10} \approx 1000$ times longer to execute than the optimized known-plaintext attack. It can be seen that the benefit of the method grows as the number of erasures increases because the method's time complexity is polynomial in the number of erasures, compared to an exponential time complexity in the case of the naive method.

For the ciphertext-only attack, the time and memory complexities are doubled, as twice as much data is required. Therefore, instead of working with $T^0_{R4_1}$ in memory, we would have to store $T^0_{R4_1}H$ (which is about twice as large). Using another approach, we can leave the required memory as in the optimized attack, and pay with higher time-complexity. We can store $T^0_{R4_1}$ in memory, and calculate the multiplication by H on the fly. This method increases the time complexity by a factor of about $e + 1$ compared to the optimized ciphertext-only attack.

6 A Passive Ciphertext-Only Cryptanalysis of A5/1 Encrypted Communication

In this section, we generalize the attack of Section 4. We show how to construct passive ciphertext-only attacks on networks that use A5/1, i.e., attacks that require the attacker to receive transmissions, but do not require the attacker to transmit. These attacks can be adapted to other ciphers, as long as the network performs error-correction before encryption.

The classic approach of implementing a ciphertext-only attack is guessing the GSM traffic (or control messages), thus, known plaintext is gained. In such a case, we can use one of the known-plaintext attacks on A5/1, as published in the literature. In this section, we discuss a different approach of implementing a ciphertext-only attack — using the fact that error-correction codes are employed before encryption. An advantage of this approach over the classic approach is that the attacker is not required to guess the contents of the traffic. The disadvantage is that the complexity of the attack is higher in the new approach.

We overview the process of the attack on A5/2 of Section 4, and generalize it. In Section 4, we constructed a function $H \cdot k$ of the keystream k. This function can be seen as a function $h(x)$ from the internal state x of the cipher at the first frame, where the internal state x determines the keystream k. The special property of this function is that it can also be efficiently computed from the ciphertext of any message that was encrypted using k, as $H \cdot k = H \cdot (C \oplus g)$, where g is a known constant. Therefore, we have a function $h(x)$ from the internal state x of the cipher, such that $h(x)$ can be also computed from the ciphertext. $h(x)$ was then reversed to reveal the internal state x (by guessing all possible $R4_1$ values, and solving a system of equations). We can find the key K_c from the internal state x by reversing the (linear) key setup.

We now follow the same lines to mount an attack in case A5/1 is used instead of A5/2. We begin by constructing the same function $h(x) : \{0,1\}^{64} \to \{0,1\}^{64}$ from the internal state of A5/1 just after the key setup (i.e., $H \cdot k$, where k is the keystream resulting from initial internal state x at the first frame). We would like to reverse $h(x) = H \cdot k$ to reveal the internal state x, knowing that the inversion of $h(x)$ is expected to be computationally intensive, as it includes inversion of A5/1. Given D data points (i.e., images under $h(x)$), it suffices to invert $h(x)$ for only one of them, as it would reveal K_c. Therefore, we treat $h(x)$ as if it is a random function, and we can use a time/memory/data tradeoff from the literature to invert it. In this discussion, we use the time/memory/data tradeoff presented by Biryukov and Shamir in [5].

Time memory tradeoffs are composed of two phases: a one-time precomputation phase and a real-time phase. The time/memory/data tradeoff in [5] has a preprocessing time complexity of N/D applications of $h(x)$, where N is the search space (2^{64} in our case), and D is the number of data points $h(x)$ that are available. The real-time phase is composed of T applications of $h(x)$ and $\sqrt{T}$ disk accesses. The attack has a good success rate (greater than 60%) when the parameters are on the tradeoff curve $TM^2D^2 = N^2$ and $D^2 \leq T \leq N$, where M is the disk space of the attacker divided by $2\log_2 N$, e.g., $M = 2^{40}$ is a $2^{40} \times 128$-bit of disk space — about 17.6 terabytes (using efficient representation, the memory complexity can drop by a factor of about 3). From the tradeoff curve, it is clear that increasing the number of available data points D by a factor of 2 reduces the time complexity of the precomputation by a factor of 2, and reduces the time complexity of the real-time phase by a factor of 4. Thus, the number of available data points is an important parameter of the attack, and the attacker benefits from having many data points.

There are a few technical issues that reduce the number of available data points of our desired form. The problem is very similar to the problem of knowing the differences between COUNT values

Table 1. Four Points on the Time/Memory/Data Tradeoff Curve for a Ciphertext-Only attack on A5/1

Attacked Channel	Available Data in Coded Messages (Four Frames)	Number of 250GBs Disks	Number of PCs to Complete Preprocessing in One Year	Duration of Online Phase on a Single PC in Minutes
KP* [6]	A Single Message	≈ 200	680	3.33
SACCH**	204 (≈ 3.5 min)	≈ 200	2800	13.33
SACCH**	600 (≈ 10 min)	≈ 200	930	1.53
SACCH**	600 (≈ 10 min)	≈ 67	930	13.83
SDCCH/8	204 (≈ 64 sec)	≈ 200	2800	13.33

* Known plaintext.
** The SACCH of the TCH/FS.

that we encounter in Section 3.3. At the time of the preprocessing, we must be able to derive the initial internal state of A5/1 over four frames (in case of SDCCH/8) from the initial internal state x in the first frame. In Section 3.3, this problem was solved by repeating the precomputation 13 times. In this section, we would not perform the precomputation several times, rather, we would wait for a data point that is covered by the precomputation, and use some other tricks.

In the rest of this section, we discuss implementations of the ciphertext-only passive attack on A5/1 under various GSM channels, and various parameters of the time/memory/data tradeoff. We compare the attacks in Table 1. Readers that are not interested in the technicalities of GSM can skip the rest of this section.

For comparison with our attacks, we analyze the time/memory/data tradeoff attack of [5] given a single known message (four frames).[4] The random function that is analyzed $h(x)$ is the function from internal state x to the 64 bits of output that are generated from x, i.e., the first bit of output is generated when the internal state is x. Thus, in a 114-bit frame, there are $114 - 64 + 1 = 51$ (overlapping) strings of 64 consecutive bits (the first 64 are at the beginning of the frame; the next 64 bits begin in the second bit of the frame, etc), with 51 internal states that are associated with them. It is enough to recover one of these internal states, as A5/1's internal state can be rolled back efficiently. As a message is transmitted over four frames, it is enough to invert $h(x)$ on one out of the $51 \cdot 4 = 204$ available 64-bit outputs of A5/1 (i.e., $D = 204$).

During the preprocessing phase, A5/1 is invoked $2^{64}/204$ times (therefore, it takes about 684 computer years, assuming 2^{22} applications of A5/1 per second can be performed on a personal computer). On a network of 1000 personal computers, the preprocessing can be completed in about eight months. Using about 50 terabytes of disk storage (200 disks of 250GBs, with $M \approx 2^{41.5}$), finding a key takes about 200 seconds of CPU time ($T \approx 2^{29.65}$), and about 30000 disk accesses (which takes less than a second when averaged on the 200 disks). Note that it is possible to reduce the number of disk accesses using A5/1's low sampling resistance (see [5, 6] for details).

We now analyze the ciphertext-only attack when employed on the SACCH of a TCH/FS and on an SDCCH/8 channel (see Appendix B for more details on these channels). We assume that $h(x)$ can be applied 2^{20} times every second on a personal computer, and that a random access to disk takes about 5 milliseconds.

[4] In Section 7 we show that it is possible to gain a known message in certain conditions.

Focus on the SACCH of a TCH/FS. In this channel, a frame is transmitted every 26 TDMA-frames, therefore, the counter $T2$ (frame number modulo 26) remains fixed. The counter $T3$ (frame number modulo 51) is increased by 26 modulo 51 with each frame of the SACCH. Note that every two frames of SACCH $T3$ is increased by one modulo 51 (as $26 \cdot 2 \equiv 1$ modulo 51).

We have to make an assumption on the frame number, such that given the internal state x of A5/1 after initialization at the first frame, we know the internal state after initialization in the other three frames of the message. We show a method that slightly loosens the assumption on the frame numbers. In the method, we use only two of the four encrypted frames. Furthermore, 20 bits of each SACCH message are fixed (the protocol requires that these bits always have the same value), therefore, we construct H with additional 20 rows, i.e., H is 292×456. While creating H, we change the order of bits in k such that $k = k_1||k_3||k_0||k_2$, where k_i are the keystreams of the individual frames (we make the corresponding changes in H's columns). Since the number of rows is 292, and due to the structure of H, we can eliminate the variables of k_1 and k_3 (i.e., $114 \cdot 2 = 228$ variables) from all the rows except for the first 228 rows by using Gauss elimination. We define the matrix H' as the rows 229–292 and columns 229–456, i.e., H' is 64×228. Using H', we define h' in a similar way to the way H defines h. Our assumption on the frame numbers is that $T1$ (the frame number divided by $26 \cdot 51 = 1326$) is the same in both the generation of k_0 and k_2, in addition we know that $T2$ remains fixed. We further assume that the value of $T3$ is even when k_0 is generated, therefore, $T3$ is larger by one in the generation of k_2 (and the two $T3$ values differ only in their LSB). These conditions are met on average about once a second. To achieve a similar tradeoff to the one given above in the BSW example, we need $D = 204$, i.e,. about three and a half minutes of conversation (since this time a single data point is four frames, compared to 51 data points in one frame in the case of known plaintext). Furthermore, the attack time, and preprocessing time is expected to take about four times longer, as the application of h' takes more CPU time than finding the output of A5/1 given an internal state. Other possible choices of parameters are given in Table 1.

Another example is the downlink SDCCH/8 channel with SACCH. In every cycle of 102 frames, three messages are transmitted for a specific phone (two SDCCH messages and one SACCH with the same error-correction code), i.e., about 6.37 messages a second. We would like to be able to calculate the XOR difference between the COUNT values in the four frames that constitute the message. Therefore, our assumption on the frame numbers is that the lower two bits of counter $T3$ are zero (this part of the assumption always holds), and that the lower two bits of counter $T2$ are zero (and the rest of the bits of $T2$ are fixed in all four frames, i.e., the counter's values (not modulo 26) in the three other frames are $T2 + 1$, $T2 + 2$, and $T2 + 3$). The assumption on $T2$ holds in six out of the 26 cases, therefore, on average the assumption holds for 1.47 messages in a second. To follow the previous tradeoff with $D = 204$, two minutes and 19 seconds are needed, which is an unreasonably long data requirement for a SDCCH/8 channel on a single session. We increase D by employing a similar trick to the one we employ in the SACCH of a TCH/FS: each GSM message can contain 184 bits, but if the message is shorter the message is padded with fill bits (whose values are publicly known) at its end. Assume that at least 20 such bits are fill bits. It's a reasonable assumption, although not always true. We perform a similar trick to the one we made for the SACCH of the TCH/FS, to construct h' from the keystream of the first two frames of the message. We modify our assumption on the frame numbers, and assume that the LSB of $T2$ is zero in the first frame, therefore, $T2$ in the second frame equals to $T2$ of the first frame with the LSB changed to 1. This assumption holds for exactly half of the possible values of $T2$, i.e., for

about $6.37/2 \approx 3.18$ messages a second. To achieve the previous tradeoff of $D = 204$, we need to collect encrypted data for a duration of about $204/(3.18) \approx 64$ seconds. The data complexity can be lowered using the tradeoff curve with a price of increased preprocessing complexity, and higher time/memory complexity. Note that the available data can be taken from several conversations, as long as they are encrypted with the same key.

7 Leveraging the Attacks to Any GSM Network by Active Attacks

In this section, we present several attacks which are based on flaws in the GSM call-establishment protocol (which is shortly described in Appendix B.1). Through these flaws, an attacker can compromise any GSM encrypted communication based on his ability to break one weak cipher of the GSM family that is supported by the victim handset. The time complexity of the new attacks is the same time complexity of breaking the weak cipher. For the sake of simplicity, we assume that the attacker wishes to compromise conversations in networks that use A5/1 through the cryptanalysis of the weaker A5/2.

Unlike the attacks of Section 4 and Section 6 which require only tapping the communications, the attacks in this section also require the attacker to transmit, and thus, the attacker takes a greater risk of being detected. However, active attacks bring many advantages to the attacker.

The major advantage that comes with the active attacks of this section is tapping into A5/1 networks with the time complexity of breaking A5/2, but there are also other advantages. In most of the active attacks that we present, the attacker impersonates the network towards the victim handset by using a fake base station. As the handset views the attacker as the network, the attacker controls the transmission power of the mobile phone, and commands it to first use high power to reduce reception errors that can cause problems during the cryptanalysis, and then use a lower power to reduce the chances of detection. Another advantage is the freedom of choosing the channel that is used, including the time slot in the TDMA frame that is allocated to the mobile. The attacker can use this freedom to reduce the complexity of the attack. For example in SDCCH/8, the uplink subchannel allocation is not as uniform as the downlink subchannel allocation. It is easier for an attacker employing a ciphertext-only attack to allocate the victim to an SDCCH/8 subchannel that he prepared for in advance (by pre-computing tables for it). The attacker can also wait a little before he commands the mobile to start encryption, such that the mobile starts encryption in a TDMA frame number that the attacker prepared for in advance (for example the attacker can precompute tables only for some values of the TDMA frame number modulo 26). For similar reasons, the attacker can also allocate a TDMA slot that is convenient to him, and he can choose the frequencies that he favors (for example, frequencies that minimize the risk of detection).

The protocol flaws that are used by the attacks are as follows:

1. The authentication and key agreement protocol can be executed between the mobile and the network at the beginning of a call, at the sole discretion of the network. The phone cannot ask for authentication. If no authentication is performed, K_c stays the same as in the previous conversation. In this case, the network can "authenticate" the phone through the fact that the phone encrypts using K_c, and thus the phone "proves" that it knows K_c.
2. The network chooses the encryption algorithm (or either not to encrypt at all).[5] The phone only reports the list of ciphers that it supports (in a message called *class-mark*).

[5] Note that if the conversation is not encrypted, a *ciphering indicator* in the phone might indicate the situation to the user.

3. The class-mark message is not protected, and can be modified by an attacker.
4. During authentication, only the phone is authenticated to the network, while there is no mechanism that authenticates the network to the phone. This fact allows for fake base-stations.[6]
5. There is no key separation: the key-agreement protocol is independent of the encryption algorithm that is used, and it is even independent of the method of communication, i.e., K_c depends only on $RAND$ (which is chosen by the network), regardless of whether A5/1, A5/2, A5/3, or even GPRS encryption algorithms is used.
6. RAND reuse is allowed: the same $RAND$ can be used as many times as the network pleases, and for different types of communications (i.e., GSM or GPRS).

7.1 Class-Mark Attack

In the simplest attack on the protocol, the attacker changes the class-mark information that the phone sends to the network at the beginning of the conversation, such that the network thinks that the phone supports only A5/2. Although the network prefers to use A5/1, it must use either A5/2 or A5/0 — no encryption, as it believes that the phone does not support A5/1. The attacker can then listen into the conversation through the cryptanalysis of the weaker A5/2 cipher.

The attacker can change the class-mark message in several ways. He can transmit his alternative class-mark message at the same time that the victim's handset transmits the class-mark message, but using a much stronger radio signal. Thus, at the cellular tower, the attacker's signal overrides the handsets original message. As an alternative, the attacker can perform a man-in-the-middle attack (enter between the handset and the cellular tower by using a fake handset and a fake base station), such that all messages pass through the attacker. Then, he can simply replace the class-mark message with another message.

Note that some networks may decide not to select A5/2, but drop the conversation. As all phones should support A5/1, this kind of attack can be easily spotted by the network, and can be prevented by insisting that the phone uses A5/1 or dropping the conversation.

7.2 Recovering K_c of Past or Future Conversations

The remaining attacks are mostly based on the fact that the protocol does not provide any key separation, i.e., the key is fixed regardless of the encryption algorithm that is used. The idea behind the attacks is to use a fake base-station[7] that instructs the phone to use A5/2, and through the attack of Section 4 on A5/2 the value of K_c is retrieved. As there is no key separation, this key is the same one used for the stronger cipher. Thus, the phone with A5/2 acts as an oracle for retrieving K_c.

In this section we present an attack in which we recover the encryption key of an encrypted conversation that was recorded in the past. As the encryption key might not change during the next few conversations (the network might choose not to perform the key-agreement protocol), the encryption key that we obtain might be valid for future conversations.

[6] It should be noted that the network "authenticates" itself to the phone through the fact that it knows how to encrypt, and thus proves knowledge of K_c. This "authentication" cannot be considered a real authentication, especially since the network can choose not to encrypt. As a result, a fake base station does not need to know the encryption key.

[7] It is easy (and cheap) to build and operate a fake base station in GSM, using off-the-shelf equipment. The fact that the phone does not authenticate the network also helps.

The simplest way of decrypting recorded conversations is when the attacker has access to the SIM card of the victim. Then, the attacker can feed the SIM card with the $RAND$ that was used in the conversation. The SIM card then calculates and returns to the attacker the respective value of K_c (this attack is possible as GSM allows re-use of $RAND$s).

Clearly, it might not be easy for the attacker to gain physical access to the victim's SIM card. Instead, the following attack simulates such an access through the use of a fake base station. As a preparation for the attack, the attacker records encrypted conversations (that may be encrypted using different K_c's). At the time of the attack, the attacker initiates a radio-session with the victim phone through the fake base station. Then, the attacker initiates an authentication procedure, using the same $RAND$ value that was used during the encrypted conversation. The phone returns $SRES$, which is equal to the $SRES$ of the recorded conversation. Next, the attacker commands the phone to start encryption using A5/2. The phone sends an acknowledgement which is already encrypted using A5/2 and the same K_c that was used in the recorded conversation (as K_c is a function of $RAND$, and the $RAND$ is identical to the one in the recorded conversation). Finally, the attacker employs the attack on A5/2 of Section 4 to obtain K_c from the encrypted response. The attack can be repeated several times for all the $RAND$s that appear in the recording.

The above attack leaves some traces, as the phone remembers the last K_c for use in the next conversation. The attacker can return the phone to its state before the attack by performing another authentication procedure using the last (legitimate) $RAND$ that was issued to the phone.

In a variation of this attack, the attacker can recover the current K_c that is stored in the phone by performing the attack, but skipping the authentication procedure. In this case, the attack does not change the state of the phone with respect to K_c. The attacker can use this K_c to tap into future conversations until the network initiates a new authentication procedure.

7.3 Man in the Middle Attack

The attacker can tap conversations in real time by performing a man-in-the-middle attack, as depicted in Figure 7.3. The attacker uses a fake base-station in its communications with the mobile phone, and impersonates the mobile phone to the network. When authentication is initiated by the network, the network sends an authentication request to the attacker, and the attacker forwards it to the victim. The victim computes $SRES$, and returns it to the attacker, which holds it and does not send it back to the network, yet. Next, the attacker asks the phone to start encryption using A5/2. This request seems legitimate to the phone, as the attacker impersonates the network. The phone starts encryption using A5/2, and sends an encrypted acknowledgment. The attacker employs the ciphertext-only attack of Section 4 to find K_c in less than a second. Only then, the attacker returns $SRES$ to the network. Now, when the attacker is "authenticated" to the network, the network asks the attacker to start encryption using A5/1. The attacker already knows K_c, and can send the response encrypted using A5/1 under the correct K_c. From this point on, the network views the attacker as the mobile phone, and the attacker can continue the conversation, relay the conversation to the mobile, etc. It should be clear that the same attack applies when using A5/3 instead of A5/1, and we note that although A5/3 can be used with key lengths of 64–128 bits, the current GSM standard only allows the use of 64-bit A5/3.

Some readers may suspect that the network may identify this attack, by identifying a small delay in the authentication procedure. However, the GSM standard allows 12 seconds for the mobile phone to complete his authentication calculations and to return an answer, while the delay incurred by this attack is less than a second.

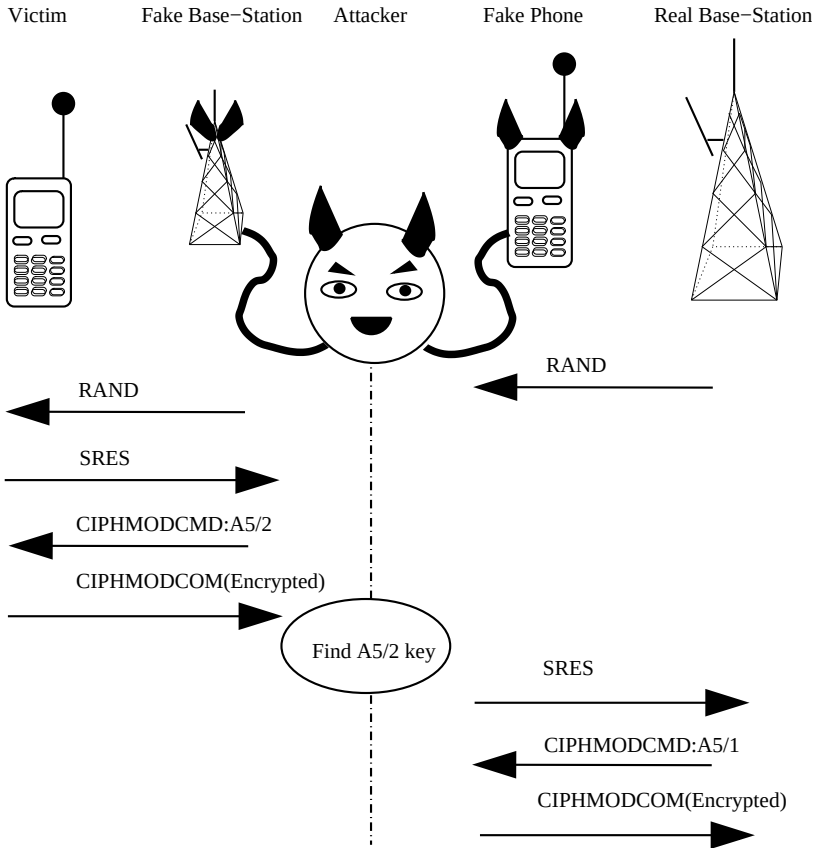

Fig. 3. The Man-in-the-Middle Attack

Another issue that might concern some readers is whether the amount of information available from the mobile suffices to mount the ciphertext only attack of Section 4. After the attacker asks the mobile to start encryption using A5/2, the mobile must reply with (an encrypted) *Cipher mode complete* (CIPHMODCOM) message, which acts as an acknowledgment that encryption has started. This message is 456 bits long (after the error-correction coding takes place). It is enough for a known-plaintext attack, but the ciphertext-only attack of Section 4 requires two such messages. Note that the attacker cannot acknowledge the CIPHMODCOM message, as he needs K_c for that. Therefore, he can wait for the retransmission mechanism of the mobile phone to transmit the encrypted CIPHMODCOM message again. Thus, the attacker obtains two differently encrypted messages, enough for the ciphertext-only attack.

It should be noted that the retransmission mechanism of GSM ensures that the CIPHMODCOM is retransmitted immediately (in the first opportunity) after the first CIPHMODCOM not acknowledged by the network, as the size of the transmission window is one. Therefore, the same message (CIPHMODCOM) is retransmitted by the mobile (but under a different frame number), and only one message bit is changed from zero to one to indicate that the message is a retransmission. As a result, not only do we gain another encrypted message, but we also gain 184 extra bits of information, which we can express as 184 extra equations for the attack of Section 4 (but we can apply the attack even without these extra equations). Moreover, for SDCCH channels, the

maximum number of retransmissions is 23. For full details on the data-link layer of GSM, we refer the reader to [14].

It appears that with a small preparation, we can infer the plaintext of the CIPHMODCOM and use the known-plaintext attack of Section 3.3. The content of the CIPHMODCOM message that the mobile returns is known or can be easily derived, except for an optional field called IMEISV. When the network asks the mobile to start encryption, it can ask that the phone's 64-bit IMEISV — International Mobile Equipment Identity (the hardware number of the phone) plus the Software Version — would be included in the CIPHMODCOM that the phone returns. If the network does not ask the phone to include the IMEISV, then the entire content of CIPHMODCOM can be inferred from the previous un-encrypted messages.

For the case that the network asks for the IMEISV, the attacker can find the IMEISV of a victim phone by some preparation. The IMEISV does not change unless the phone is replaced, or its software is upgraded. In the preparation work, the attacker can ask the mobile (through a fake base station) not to encrypt, but to include its IMEISV. Thus he gains the IMEISV, and in future attacks he can employ the known-plaintext attack of Section 3.3. Alternatively, the attacker can ask the mobile to encrypt, but not to include the IMEISV, and employ the known-plaintext attack to find K_c. Then, the attacker releases the connection, and initiates a new connection skipping the authentication, this time the attacker asks the mobile to encrypt using A5/2 and to include the IMEISV. Since K_c is known from the previous section, the attacker gains the IMEISV for future attacks. It should be noted that the known plaintext that is achieved through guessing the CIPHMODCOM can be used for attacks on other GSM ciphers, such as A5/1. For a full description of the CIPHMODCOM message, see [13].

A possible pitfall of the attack is that some networks employ protective measures that spot the event that two radio sessions are maintained from a single identity. This event implies that the phone has been cloned, and the network freezes the subscriber's account. This kind of event might occur during the establishment of a man-in-the-middle attack, when the attacker impersonates the phone to the network, but lost the acquisition on the mobile victim, which holds another radio-session. It is very easy to avoid this event if the attacker identifies (as the victim) to the network, only after he has an active radio-session with the victim. The GSM protocol also allows the attacker to prevent the mobile from accessing (non-faked) base station, by noting to the mobile that there are no other base stations except the faked one.

7.4 Attack on GPRS

GPRS can be attacked by an active attack, due to the fact that there is no key separation between voice conversation and GPRS data, even if the ciphers used in GPRS are secure. For example, the attacker can listen in to the $GPRS\text{-}RAND$ sent by the network to the handset, while impersonating the voice network towards the handset.[8] Then, the attacker initiates a radio session on the voice network with the handset and performs the attack that retrieves the K_c using $RAND = GPRS\text{-}RAND$. As GPRS uses the same SIM (with the same algorithms and without any key separation from regular GSM), K_c equals $GPRS\text{-}K_c$. The attacker can now decrypt/encrypt the customer's GPRS traffic using the recovered K_c. Alternatively, the attacker can record the customer's traffic, and perform the impersonation at any later time to retrieve the $GPRS\text{-}K_c$. Then, the recorded data can be decrypted. It is rumored that the first two GPRS encryption algorithms (which are

[8] The handset can work with one cellular tower for regular GSM, and another cellular tower for GPRS.

kept in secret) are weaker than the newer ones. If indeed they are weak, it is also possible to mount the attack the other way round, finding $GPRS\text{-}K_c$, and using it to decrypt voice communication.

8 Possible Attack Scenarios

The attacks presented in this paper can be used in several scenarios. In this section, we present four of the scenarios: call wire-tapping, call hijacking, altering of data messages (SMS), and call theft — dynamic cloning.

8.1 Call Wire-Tapping

The most naive scenario that one might anticipate is eavesdropping conversations in real-time. Communications encrypted using GSM can be decrypted and eavesdropped by an attacker, once the attacker has the encryption key. Not only can the attacker tap voice conversation, but he can also tap data conversations and SMS messages. The attacker can tap video and picture messages that are sent over GPRS, etc. Real-time eavesdropping on A5/2 networks can be performed using a passive attack on A5/2 as shown in Section 4. On networks using encryption other than A5/2, the man-in-the-middle attack of Section 7 is required, or the passive attack of Section 6 can be used (but with a very long precomputation, and a very large storage).

In another possible wire-tapping attack against ciphers such as A5/1, the attacker records the encrypted conversation (making sure that he knows the $RAND$ value that that is sent unencrypted). Then, he uses a fake base station to attack the victim phone and retrieve the respective K_c. Once the attacker has the key, he simply decrypts the conversation. Note that an attacker can record many conversations, and with subsequent later attacks recover all the keys. This attack has the advantage of transmitting only in the time that is convenient for the attacker. Possibly even years after the recording of the conversation, or when the victim is in another country, or in a convenient place for the attacker.

8.2 Call Hijacking

While a GSM network can perform authentication at the initiation of the call, encryption is the means of GSM for preventing impersonation at later stages of the conversation. The underlying assumption is that an imposter does not have K_c, and thus cannot conduct encrypted communications. Using our passive attacks, the attacker can obtain the encryption key. Once an attacker has the encryption key, he can cut the victim off the conversation (by transmitting a stronger signal, for example), and impersonate the victim to the other party using the retrieved key. Therefore, hijacking the conversation after authentication is possible.

Hijacking can occur during early call-setup, even before the victim's phone begins to ring. The operator can hardly suspect that an attack is performed. The only clue of an attack is a moment of some increased electro-magnetic interference.

In another way of call hijacking, the attacker mounts the man-in-the-middle attack. Then, at any point in time (even before the phone rings), the attacker can disconnect the victim handset and take over the conversation (including forwarding the conversation to another location).

8.3 Altering of Data Messages (SMS)

Once a call has been hijacked, the attacker decides on the content, including on the content of SMS messages (which are encrypted by the same K_c as the speech). The attacker can eavesdrop on the contents of a data message being sent by the victim (or being received), and send his own version instead. The attacker can also stop the message from being received, or even send his own SMS message, thus compromising the integrity of GSM traffic.

8.4 Call Theft — Dynamic Cloning

GSM was believed to be secure against call theft due to the authentication procedures of A3A8 (at least for operators that use a strong primitive for A3A8 rather then COMP128).

However, due to the weaknesses discussed in this paper, an attacker can make outgoing calls on the expense of a victim. When the network asks for authentication, the attacker performs the attack that uses the victim's phone as an oracle for obtaining the $SRES$ and K_c for the given $RAND$ (as described in Section 7): the attacker initiates an outgoing call to the cellular network in parallel to a radio session to a victim. When the network asks the attacker for authentication, the attacker asks the victim for authentication, and relays the resulting authentication back to the network. The attacker then recovers K_c as described in Section 7. Now the attacker can close the session with the victim, and continue the outgoing call to the network. This attack is hardly detectable by the network, as the network views it as normal access. The victim's phone does not ring, and the victim has no indication that he is a victim (until his monthly bill arrives).

9 How to Acquire a Specific Victim

We distinguish between attacks that are targeted against a specific victim (e.g., eavesdropping), and attacks that are not targeted against a specific victim (e.g., call-theft). When performing eavesdropping, the attacker is usually interested in a specific victim which he targets. However, in call theft, the attacker's aim is to steal calls, and he does not care whether victim A pays the bill, or victim B pays the bill, as long as the attacker does not pay. This section focuses on targeting a specific victim.

GSM includes a mechanism that is intended to provide protection on the identity of the mobile phone. Each subscriber is allocated a TMSI (Temporary Mobile Subscriber Identity) over an encrypted link. The TMSI can be reallocated every once in a while, in particular when the subscriber changes his location. The TMSI is used to page the subscriber on incoming calls and for identification during the un-encrypted part of a session. On first sight, it seems that an attacker that performs eavesdropping with cryptanalysis using one of the methods of the previous sections can follow the decrypted data, and obtain the TMSI of his targeted victim. However, the fixed identification of a mobile is its International Mobile Subscriber Identity (IMSI), which might be unknown to the attacker. If both the IMSI and TMSI are unknown to the attacker, he may be forced to listen in to all the conversations in the area until he recognizes the victim's voice.

The attacker might only have the victim's phone number, and wish to associate the phone number with the subscriber's IMSI or TMSI. There are several possible solutions to this problem: In one solution the attacker calls the victim's phone, and pretends it to be a mistake in dialing. By monitoring all communications in the area the attacker can distinguish the victim's phone, by recognizing his own caller ID, for example. Another more covert solution is to send a malformed

SMS message to the target phone. For example, the attacker can send an SMS message as if it is part of a multi-part SMS message, but actually send only one part of the SMS. This part is received in the victim's phone, but since the entire SMS message is never fully received, the phone does not indicate to the user of the received SMS. However, the SMS passes through the radio-interface, and thus the victim can be identified. This solution can also be used as a source of known-plaintext, even during a call (when an SMS is transmitted during a call on a voice channel, an un-encrypted flag signals that data is transmitted instead of voice. If the SMS is transmitted on the SACCH, the attacker would have to guess on which bursts the SMS is carried). The attacker might be successful in identifying the victim's TMSI by correlating the paging information on the serving base station with, for example, the SMS that the attacker sends.

When performing an active attack, the attacker needs to lure the mobile into his own (fake) base station. The luring is accomplished by a suitable choice of the parameters of the fake base station, causing the victim mobile to prefer the attacker's base station. However, the fake base station might lure "innocent" handsets in addition to the victim handset. Therefore, the acquisition is composed of four phases:

1. luring many mobiles including the victim,
2. sensing the victim,
3. isolating the victim, and
4. returning the "innocent" mobiles back to the original network.

The sensing of the victim can be performed in a few ways. One way to sense the victim is to set a parameter called the *location area* of the fake base station to be different than the surrounding legitimate base stations. Once lured, the mobile has to perform a procedure called *location area update*, which includes contacting the fake base station and identifying (a mobile must perform location area update when switching between base stations with different values of the location area parameter). Another way (assuming the TMSI or the IMSI is known) is to use the same location area, and to page the victim in the fake base station using its TMSI/IMSI until the victim responds (once the victim handset is parked on the fake base station, it must respond). If the TMSI/IMSI is not known, the attacker can use the radio-session of the location area update to interrogate the mobile for its IMSI (if only the TMSI is known), or to perform an acquisition as previously described. The attacker can relay the paging messages of the real network to the lured mobiles, so they do not miss incoming calls.

The next steps for the attacker are to isolate the victim and return the "innocent" handsets to the real network. The isolation can be performed by changing the fake base station parameters, such that it transmits on its beacon frequency that the fake base station is the only cell in the area. This change prevents the lured mobiles from switching to other base stations. The attacker can now page the victim to make sure that the victim is still parked on the fake base station.

Next, the attacker returns the "innocent" handsets back to the real network by initiating a radio-session with each one of them, and returning them to the real network: During the radio session, the handsets are made to believe that they are handed-over to a neighbor base station, while actually the attacker uses another transceiver (fake base station without the beacon frequency) to impersonate that neighbor base station. After the "handover" is complete, the radio-session is released, and the "innocent" mobile returns to the real neighbor base station. In another option for returning innocent mobiles to the real network, the attacker establishes a radio-session with the victim, and "scares away" all the other mobiles, for example by stopping transmission on the beacon frequency. After a short time, the beacon can be restored with parameters that are unlikely

to attract mobiles, but claiming to be the only base station in the area. Before releasing the radio-session with the victim, the victim is handed over to the fake base station with the new parameters. Accidental entrance of other mobiles to the base station can be identified using a different location area for the fake base station, and a radio session can then be established with these mobiles, during which they are returned to the real network. It is stressed that a correct choice of parameters for the fake-base station should almost entirely eliminate accidental entries to the base station.

10 Summary

In this paper, we present new methods for attacking the encryption and the security protocols used by GSM and GPRS. The described attacks are easy to apply, and do not require knowledge of the conversation. We stress that GSM operators should replace the cryptographic algorithms and protocols as soon as possible, or switch to the more secure third generation cellular system (although it still possesses some of the weaknesses described in this paper).

Even GSM networks that use the new A5/3 succumb to our attacks. We suggest to change the way A5/3 is integrated into GSM, in order to protect the networks from such attacks. A possible correction is to make the keys used in A5/1 and A5/2 unrelated to the keys that are used in A5/3. The integration of GPRS suffers from similar flaws that should be taken into consideration.

We would like to emphasize that our ciphertext-only attack is made possible by the fact that the error-correction codes are employed before the encryption. In the case of GSM, the addition of such a structured redundancy before encryption is performed crucially reduces the security of the system.

As a result of the initial publication of these attacks, the GSM association security group together with the GSM security working group are working to remove the A5/2 algorithm from handsets.

Acknowledgements

We are grateful to Orr Dunkelman for his great help and various comments on early versions of this work, and to Adi Shamir for his advice and useful remarks. We would like to thank David Wagner for providing us with information on his group's attack on A5/2. We also acknowledge the anonymous referees for their important comments. Finally, we would like to thank the many people that expressed their interest in this work.

References

1. *The 3rd Generation Partnership Project (3GPP)*, `http://www.3gpp.org/`.
2. Elad Barkan, Eli Biham, *Conditional Estimators: an Effective Attack on A5/1*, proceedings of SAC 2005, LNCS 3897, pp. 1–19, Springer-Verlag, 2006.
3. Elad Barkan, Eli Biham, Nathan Keller, *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communications*, Advances in Cryptology, proceedings of Crypto 2003, Lecture Notes in Computer Science 2729, Springer-Verlag, pp. 600–616, 2003.
4. Eli Biham, Orr Dunkelman, *Cryptanalysis of the A5/1 GSM Stream Cipher*, Progress in Cryptology, proceedings of Indocrypt'00, Lecture Notes in Computer Science 1977, Springer-Verlag, pp. 43–51, 2000.
5. Alex Biryukov, Adi Shamir, *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*, Advances in Cryptology, proceedings of Asiacrypt 2000, Lecture Notes in Computer Science 1976, Springer-Verlag, pp. 1–13, 2000.

6. Alex Biryukov, Adi Shamir, David Wagner, *Real Time Cryptanalysis of A5/1 on a PC*, Advances in Cryptology, proceedings of Fast Software Encryption'00, Lecture Notes in Computer Science 1978, Springer-Verlag, pp. 1–18, 2001.
7. Marc Briceno, Ian Goldberg, David Wagner, *A pedagogical implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms*, `http://cryptome.org/gsm-a512.htm` (originally on www.scard.org), 1999.
8. Marc Briceno, Ian Goldberg, David Wagner, *An implementation of the GSM A3A8 algorithm*, `http://www.iol.ie/~kooltek/a3a8.txt`, 1998.
9. Marc Briceno, Ian Goldberg, David Wagner, *GSM Cloning*, `http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html`, 1998.
10. Nicolas Courtois, Alexander Klimov, Jacques Patarin, Adi Shamir, *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*, Advances in Cryptology, proceedings of Eurocrypt 2000, Lecture Notes in Computer Science 1807, Springer-Verlag, pp. 392–407, 2000.
11. Patrik Ekdahl, Thomas Johansson, *Another Attack on A5/1*, IEEE Transactions on Information Theory 49(1), pp. 284-289, 2003.
12. European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Channel Coding*, TS 100 909 (GSM 05.03), `http://www.etsi.org`.
13. European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Mobile radio interface; Layer 3 specification*, TS 100 940 (GSM 04.08), `http://www.etsi.org`.
14. European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Mobile Station - Base Stations System (MS - BSS) Interface Data Link (DL) Layer Specification*, TS 100 938 (GSM 04.06), `http://www.etsi.org`.
15. European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Multiplexing and multiple access on the radio path*, TS 100 908 (GSM 05.02), `http://www.etsi.org`.
16. European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Physical layer on the radio path; General description*, TS 100 573 (GSM 05.01), `http://www.etsi.org`.
17. European Telecommunications Standards Institute (ETSI), *Digital cellular telecommunications system (Phase 2+); Security related network functions*, TS 100 929 (GSM 03.20), `http://www.etsi.org`.
18. Ian Goldberg, David Wagner, Lucky Green, *The (Real-Time) Cryptanalysis of A5/2*, presented at the Rump Session of Crypto'99, 1999.
19. Jovan Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, Advances in Cryptology, proceedings of Eurocrypt '97, Lecture Notes in Computer Science 1233, pp. 239–255, Springer-Verlag,1997.
20. Alexander Maximov, Thomas Johansson, Steve Babbage, *An improved correlation attack on A5/1*, proceedings of SAC 2004, LNCS 3357, pp. 1–18, Springer-Verlag, 2005.
21. Security Algorithms Group of Experts (SAGE), *Report on the specification and evaluation of the GSM cipher algorithm A5/2*, `http://cryptome.org/espy/ETR278e01p.pdf`, 1996.
22. Slobodan Petrović, Amparo Fúster-Sabater, *Cryptanalysis of the A5/2 Algorithm*, IACR ePrint Report 2000/052, `http://eprint.iacr.org`, 2000.

A Enhancing The Attack of Goldberg, Wagner, and Green on GSM's A5/2 to a Ciphertext-Only Attack

We now describe a ciphertext-only attack on A5/2 based on Goldberg, Wagner, and Green's Attack [18]. We use the same matrix H as in Section 4. Recall that the attack of [18] requires the XOR difference of the keystream of two frames. The enhanced ciphertext-only attack uses eight encrypted frames. We denote the eight encrypted frames by $C_1, \ldots, C_8$, where the first four frames have consecutive frame numbers fr_1, fr_2, fr_3, fr_4, and the second four frames have consecutive frame numbers fr_5, fr_6, fr_7, fr_8. We require that fr_{i+4} is exactly $51 \cdot 26 = 1326$ frames after fr_i, for $i \in \{1, 2, 3, 4\}$. We also require that $f_1/1326$ is even (required by the original attack), and that $C_i, C_{i+1}, C_{i+2}, C_{i+3}$, where $i \in \{1, 5\}$, constitute an encrypted message. The latter requirement does not hold for the SACCH of the TCH/FS, due to the locations of TDMA frame numbers that can

be used to transmit a SACCH message, however, it holds for the SDCCH/8 channel (an adjusted requirement can be constructed for other channels, including the TCH/FS).

Due to the reasons shown in Section 4, it holds that

$$H \cdot (C_1 \oplus g||C_2 \oplus g||C_3 \oplus g||C_4 \oplus g) = H \cdot (k_1||k_2||k_3||k_4),$$

where k_i is the keystream used in frame f_i. Similarly it holds that

$$H \cdot (C_5 \oplus g||C_6 \oplus g||C_7 \oplus g||C_8 \oplus g) = H \cdot (k_5||k_6||k_7||k_8).$$

Due to linearity, it holds that:

$$H \cdot ((C_1||C_2||C_3||C_4) \oplus (C_5||C_6||C_7||C_8)) = \\ H \cdot ((k_1||k_2||k_3||k_4) \oplus (k_5||k_6||k_7||k_8)).$$

Let

$$C' = (C_1||C_2||C_3||C_4) \oplus (C_5||C_6||C_7||C_8),$$

and let

$$k' = (k_1||k_2||k_3||k_4) \oplus (k_5||k_6||k_7||k_8).$$

Therefore, $HC' = Hk'$.

The rest of the attack is similar to the attack of [18], using $Hk' = HC'$ instead of the keystream difference. Using a similar argument to the one in Section 3.1 and given the initial value of $R4_1$, we express the bits of the 272-bit $H \cdot C'$ as linear expressions of the bits of the initial value of $R1_1$, $R2_1$, and $R3_1$ at the first frame. The flaw observed in [18] causes $R4$ to have the same value in fr_i and fr_{i+4}, where $i \in \{1, 5\}$. Thus, the clockings are the same in these frames, and each bit of k_i and k_{i+4} can be expressed using exactly the same quadratic terms over the bits of $R1$, $R2$, and $R3$. The XOR difference of these terms is linear in the bits of $R1$, $R2$, and $R3$. To further simplify the analysis, we assume that the XOR difference among the frame numbers is known in advance. Since the difference between the frame numbers is known, a guess for a value for $R4$ of the first frame causes a known value for $R4$ of the other frames. In addition, the respective differences between the values of registers $R1$, $R2$, and $R3$ in the four frames are also known in advance. In this way, we can express Hk' as linear terms. It should be noted that we do not have to use the whole 272 bits of $H \cdot C'$, and actually less than a hundred bits suffices.

The attack follows a similar path as the original attack, using the redundancy to filter wrong $R4$ values. The time complexity of this attack is similar to the one of the original attack (i.e., a few milliseconds on a personal computer), and the memory requirement is also similar, i.e., about 15 MBs of volatile memory and another 60 MBs of memory that can be stored on disk. The pre-computation takes similar time. The time complexity of this enhanced attack is better than the ciphertext-only attack of Section 4, however, the fact that fr_5 should be exactly 1326 frames after fr_1 (about six seconds) limits the usability of this attack compared to the one in Section 4, which can complete in less than a second given eight encrypted frames.

B Technical Background on GSM

In this appendix we describe some technical aspects of the GSM system, which are relevant to attacks presented in this paper.

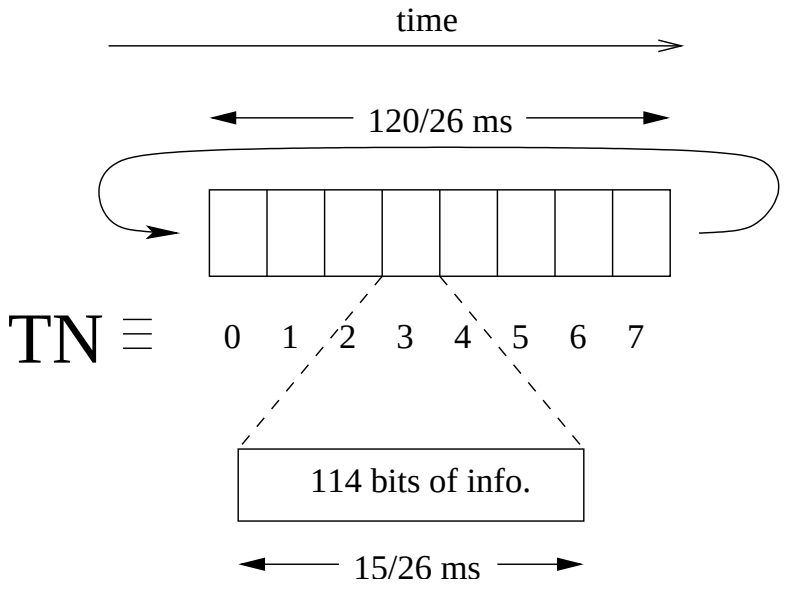

Fig. 4. A TDMA frame

We first elaborate on the concept of a TDMA frame. In GSM the same physical channel can serve up to eight different phones, by allocating the physical channel to different phones through round-robin, where each phone transmits in a *time slot* that lasts 15/26 ms. This method is known as *Time Division Multiple Access* (TDMA). Each frame is composed of eight time slots, which are referred to by their *Time slot Number* (TN). In Figure 4 we depict a typical TDMA frame. Each TDMA frame has a TDMA frame number associated with it. The TDMA frame number is fixed for all the time slots in the TDMA frame, and is incremented by one before the next TDMA frame begins. In each time slot, 114 bits of information can be transmitted. Therefore, the physical channel between the network and a phone has a maximum throughput of 114 bits per TDMA frame, or 24.7 Kbits/second.[9] In this paper, we always focus on the link between a single phone and the network, and therefore, when referring to a frame we refer to the data in the relevant slot for the phone in the TDMA frame.

The keystream generation (using A5) for a specific frame depends on the TDMA frame number. In Section 2, we describe the way that COUNT affects the A5 key setup. COUNT is derived from the TDMA frame number as shown in Figure 5, where $T1$ is the quotient of the frame number divided by $51 \cdot 26 = 1326$, $T2$ is the remainder of the frame number divided by 26, and $T3$ is the remainder of the frame number divided by 51. It should be noted that many times in our attacks, we know in advance the additive difference between two frame numbers, but we do not know in advance (with 100% certainty) the XOR-difference between the COUNT values of the two frames. This fact complicates our attack at certain points. Note that the above description is true only when the mobile is allocated a single time slot. When the mobile is allocated several time slots (or in GPRS), a different method is used.

[9] Note that the actual throughput is lower due to error-correction codes that must be employed, protocols overhead, and the fact that several logical channels between the phone and the network share the same physical channel. In GPRS, a higher data rate is accomplished by allocating several time slots to the same phone.

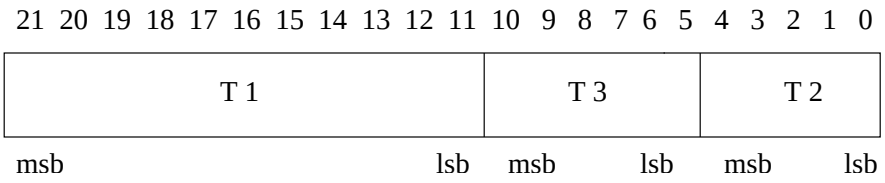

Fig. 5. The coding of COUNT

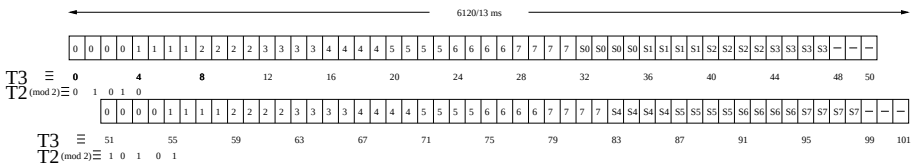

Fig. 6. The SDCCH/8 channel — downlink.

There are many kinds of messages in GSM, but most of them consume 456 bits after error correction. The allocation of the 456-bit message into frames depends on the channels. Here are two extreme examples: the 456-bit message is transmitted on four consecutive frames in some channels, but there is also a channel in which the 456-bit message is transmitted over 22 frames (interleaved with other messages). In the following paragraphs, we give two examples of two specific channels. For exact description of GSM channels see [16].

The slowest dedicated channel in GSM is a Stand alone Dedicated Control CHannel (SDCCH/8), which is used mostly for signaling in the beginning of a call, or for SMS transfer (while not in a voice conversation). In this channel, the same TN is used by up to eight different mobiles, i.e., the SDCCH contains eight *subchannels* $0, \ldots, 7$. The subchannel is determined by the value of $T3$ and the LSB of $T2$. Each mobile is also allocated a Slow Associated Control CHannel (SACCH). The downlink (from the network to the mobile) frame arrangement is shown in Figure 6, where a number "x" denotes messages belonging to a SDCCH subchannel x, Sx denotes the SACCH of subchannel x, and an empty frame is denoted by "–". Each 456-bit message is transmitted in four consecutive frames. When $T3 \equiv 48, 49$, or 50 no frames are transmitted. The uplink frame arrangement of SDCCH/8 is shown in Figure 7.

Another highly-used channel in GSM is the full rate traffic channel for speech (TCH/FS), which is used to carry voice conversations. In this channel, the 456-bit speech message are transmitted on eight frames, using the even-numbered bits of the first four frames, and the odd-numbered bits of the second four frames (the remaining bits carry parts of the previous and next speech messages).

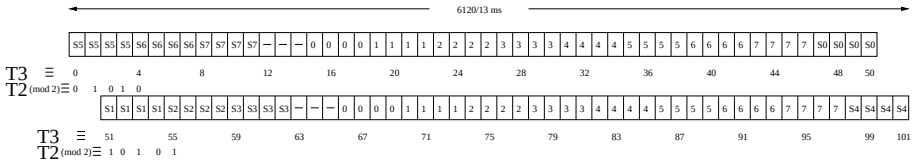

Fig. 7. The SDCCH/8 channel — uplink.

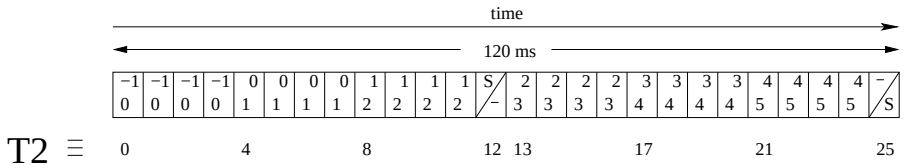

Fig. 8. The TCH/FS.

Each mobile in TCH/FS is also allocated a SACCH channel, as shown in Figure 8, where a SACCH frame is denoted by "S", a number inside a frame denotes a speech message (the value at the top of an entry denotes a speech message carried on odd-numbered bits, and the value at the bottom of an entry denotes a speech message carried on even-numbered bits), and an empty frame is denoted by "–". In each period of $T2$ one SACCH frame is transmitted, either when $T2$ is 12 or when $T2$ is 25 (using both the even-numbered bits and the odd-numbered bits), and the other frame (when $T2$ is 25 or 12, respectively) is left empty. The choice of the frame in which the SACCH is transmitted depends on the LSB of the TN that is allocated to the mobile (when the LSB is zero the SACCH is transmitted when $T2$ is 12). A 456-bit SACCH message starts whenever the TDMA frame number modulo 104 equals $12+13 \cdot TN$. For further details on the TDMA frame number in which a message can begin, see [15].

There are many types of channels, the above are only a few examples.

B.1 GSM Call Establishment

Calls in GSM are established as follows:

1. (In case the call is initiated by the network:) The network pages the phone with PAGING REQUEST by its IMSI or TMSI on the cell's paging channel (PAGCH). The configuration of the PAGCH is a part of a cell's broadcast information. If the call is initiated by the mobile it starts directly from stage 2.
2. Immediate assignment procedure[10]:
   (a) The phone sends a CHANNEL REQUEST message on the random access channel (RACH). The CHANNEL REQUEST message includes a very small amount of information — only 8 bits. It does not contain an identification of the mobile, rather it includes a random discriminator (5 bits). The remaining three bits contain the establishment cause.
   (b) The network broadcasts an IMMEDIATE ASSIGNMENT message on the PAGCH. This message contains the random discriminator (and also the TDMA frame number in which the CHANNEL REQUEST was received), and the details of the channel that is allocated to the mobile (including frequency hopping information, if needed). The message also includes other technical information such as timing advance. The mobile immediately tunes to the assigned traffic channel.[11]
3. Service Request and Contention Resolution:

[10] The procedure is initiated by the mobile phone. It can be triggered by a PAGING REQUEST, or by a service request originated by the mobile.

[11] Unlike the PAGCH and the RACH which are uni-directional, a traffic channel is a bi-directional channel

(a) The mobile sends a service request message (e.g., paging response, service request, etc.), this message includes the TMSI of the mobile. The message also includes the mobile classmark (including the A5 versions that are supported), and a ciphering key sequence number $(0, \ldots, 6)$.

(b) The network acknowledges the service request message, and repeats the TMSI. The reason for repeating the TMSI is contention resolution: It is possible that two mobiles used the same random discriminator on the same TDMA frame, and therefore, both "think" that they are assigned to the same channel. The mobile that his TMSI is acknowledged by the network, stays on the channel, and the other mobile quits.

4. Authentication:[12]
   (a) The network sends authentication request (AUTHREQ). The authentication request includes a random 128-bit value RAND, and a ciphering key sequence number, in which the resulting K_c should be stored.
   (b) The mobile answers the authentication with the computed signed response (SRES), in an authentication response message (AUTHRES).
   (c) The network asks the mobile to start encryption using a cipher mode command (CIPHMODCMD). The network can specify the encryption algorithm to be used, and it specifies the encryption key by a ciphering key sequence number $(0, \ldots, 6)$. The network starts to decipher incoming communication. This message can also be used to ask the mobile to send its international mobile equipment identity, and software version (IMEISV).
   (d) The mobile starts to encrypt and decrypt, and responds with (encrypted) cipher mod complete message (CIPHMODCOM). If requested, the mobile sends its IMEISV.
5. The network and the mobile "talk" on the channel. It might well be that the network changes the channel. For example, if it is a voice conversation the channel might need to be changed to suit a voice conversation, etc. In case a channel is changed or a handover is needed, the new channel information is sent by the network (including the frequency hopping information). Note that if the conversation is encrypted, then the new channel information is encrypted as well.

It is important to understand the concept of traffic channels in GSM. A traffic channel in GSM is composed of a list of frequencies, and frequency hopping parameters: Mobile Allocation Index Offset (MAIO), which takes a value from zero to the number of frequencies in the list minus one, and the Hopping Sequence Number (HSN), which takes a value from zero to 63. Therefore, given n frequencies there are $64n$ different hopping sequences. Usually, traffic channels in the same cell bear the same HSN and different MAIOs. After a traffic channel is assigned, the mobile and the network compute the frequency for each burst according to the above information given at the time of assignment, and according to the TDMA frame number (which is publicly known). The channel remains the same one even when encryption is turned on. The channel may be changed during the course of the conversation. In this case, the new channel parameters are passed on the current channel.

[12] The network can choose to perform authentication every call, but may also choose to skip this procedure (and use an already existing K_c for encryption, or choose not to encrypt).