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] Department of Mathematics
Technion – Israel Institute of Technology
Haifa 32000, Israel
Email: nkeller@tx.technion.ac.il

Abstract. In this paper we present a very practical ciphertext-only cryptanalysis of GSM encrypted communication, and various active attacks on the GSM protocols. These attacks can even break into GSM networks that use "unbreakable" ciphers. We 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 then extend this attack to a (more complex) ciphertext-only attack on A5/1. We describe new attacks on the protocols of networks that use A5/1, A5/3, or even GPRS. These attacks are based on security flaws of the GSM protocols, and work whenever the mobile phone supports A5/2. We emphasize that these attacks are on the protocols, and are thus applicable whenever the cellular phone supports a weak cipher, for instance they are also applicable 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. These attacks allow attackers to tap conversations and decrypt them either in real-time, or at any later time. We also show active attacks, such as call hijacking, altering of data messages and call theft.

1 Introduction

GSM is the most widely used cellular technology. By December 2002, more than 787.5 million GSM customers in over 191 countries formed approximately 71% of the total digital wireless market. GPRS (General Packet Radio Service) is a new service for GSM networks that offer 'always-on', higher capacity, Internet-based content and packet-based data services. It enables services such as color Internet browsing, e-mail on the move, powerful visual communications, multimedia messages and location-based services.

GSM incorporates security mechanisms. Network operators and their customers rely on these mechanisms for the privacy of their calls and for the integrity of the cellular network. The security mechanisms protect the network by authenticating customers to the network, and provide privacy for the customers by encrypting the conversations while transmitted over the air.

There are three main types of cryptographic algorithms used in GSM: A5 is a stream-cipher used for encryption, A3 is an authentication algorithm and A8 is the key agreement algorithm. The design of A3 and A8 is not specified in the specifications of GSM, only the external interface of these algorithms is specified. The exact design of the algorithm can be selected by the operators independently. However, many operators used the example, called *COMP128*, given in the GSM memorandum of understanding (MoU). Although never officially published, its description was found by Briceno, Goldberg, and Wagner [6]. They have performed cryptanalysis of COMP128 [7], allowing to find the shared (master) key of the mobile phone and the network, thus allowing cloning. The description of A5 is part of the specifications of GSM, but it was never made public. There are two currently used versions of A5: A5/1 is the "strong" export-limited version, and A5/2 is the "weak" version that has no export limitations. The exact design of both A5/1 and A5/2 was reverse engineered by Briceno [5] from an actual GSM telephone in 1999 and checked against known test-vectors. An additional new version, which is standardized but not yet used in GSM networks is A5/3. It was recently chosen, and is based on the block-cipher KASUMI. We note that a similar construction based on KASUMI is also used in third generation networks (3GPP) [1], on which we make no claims in this paper.

A5/1 was initially cryptanalized by Golic [14], and later by: Biryukov, Shamir and Wagner [4], Biham and Dunkelman [2], and recently by Ekdahl and Johansson [11].

After A5/2 was reverse engineered, it was immediately cryptanalized by Goldberg, Wagner and Green [13]. Their attack is a known plaintext attack that requires the difference in the plaintext of two GSM frames, which are exactly 2^{11} frames apart (about 6 seconds apart). The average time complexity of this attack is approximately 2^{16} dot products of 114-bit vectors.[3] A later work by Petrović and Fúster-Sabater [17] suggests to treat the initial internal state of the cipher as variables, write every output bit of A5/2 as a quadratic function of these variables, and linearize the quadratic terms. They showed that the output of A5/2 can be predicted with extremely high probability after a few hundreds of known output bits. However, this attack does not discover the initial key of A5/2. Thus, it is not possible to use this attack as a building block for more advanced attacks, like those presented in Section 5. This attack's time complexity is proportional to 2^{17} Gauss eliminations of matrices of size of about 400×719.[4] This latter attack actually uses overdefined systems of quadratic

[3] We observe that this attack [13] is not applicable (or fails) in about half of the cases, since in the first frame it requires that after the initialization of the cipher the 11th bit of R4 is zero.

[4] The paper [17] does not clearly state the complexity. The above figure is our estimate.

equations during its computations, however, there is no need to solve the equations to perform cryptanalysis — the equations are only used to predict the output of future frames.

Solving overdefined systems of quadratic equations — as a method of cryptanalysis drew significant attention in the literature. This method has been applied initially by Kipnis and Shamir to the HFE public key cryptosystem in [15], later improved by Courtois, Klimov, Patarin, and Shamir in [9]. Further work include: Courtois and Pieprzyk's cryptanalysis of block ciphers [10]. The method has also been applied to stream ciphers, see Courtois work on Toyocrypt [8]. The complexity of most of these methods seems difficult to evaluate.

In this paper we show how to mount a ciphertext-only attack on A5/2. This attack requires a few dozen milliseconds of encrypted data, and its time complexity is about 2^{16} dot products. In simulations we made, our attack found the key in less than a second on a personal computer. We show that the attack we propose on A5/2 can be leveraged to mount an active attack even on GSM networks that use A5/1, A5/3, or GPRS networks, thus, realizing a real-time active attack on GSM networks, without any prior required knowledge. The full attack is composed of three main steps:

1. The first step is a very efficient known plaintext attack on A5/2 that recovers the initial key. This first attack is algebraic in nature. It takes advantage of the low algebraic order of the A5/2 output function. We represent the output of A5/2 as a quadratic multivariate function in the initial state of the registers. Then, we construct an overdefined system of quadratic equations that expresses the key-stream generation process and we solve the equations.
2. The second step is improving the known plaintext attack to a ciphertext-only attack on A5/2. We observe that GSM employs Error-Correction codes before encryption. We show how to use this observation to adapt the attack to a ciphertext-only attack on A5/2.
3. The third step is leveraging of an attack on A5/2 to an active attack on A5/1, A5/3, or GPRS-enabled GSM networks. We observe that due to the GSM security modules interface design, the key that is used in A5/2 is the same as in A5/1, A5/3, and GPRS. We show how to mount an active attack on any GSM network.

We then show how to mount a passive ciphertext-only attack on networks that employ A5/1. It is basically a time/memory/data tradeoff attack. There are many choices for the parameters of the attack, four of which are given in Table 1. This attack on A5/1 can be similarly leveraged to active attacks on the protocols of GSM, but the complexity is higher than the A5/2 case.

This paper is organized as follows: In Section 2 we describe A5/2, and the way it is used, and give some background on GSM security. We present our new known plaintext attack in Section 3. In Section 4 we improve our attack to a ciphertext-only attack. In Section 5 we show how to leverage the ciphertext-only attack on A5/2 to an active attack on any GSM network. We then describe the passive ciphertext-only attack on A5/1 in Section 6. We discuss the implications of the attacks under several attack scenarios in Section 7. Section 8 summarizes the paper.

2 Description of A5/2 and GSM Security Background

In this section we describe the internal structure of A5/2 and the way it is used. A5/2 consists of 4 maximal-length LFSRs: R1, R2, R3, and R4. These registers are of length 19-bit, 22-bit, 23-bit, and 17-bit respectively. Each register has taps and a feedback function. Their irreducible polynomials are: $x^{19} \oplus x^5 \oplus x^2 \oplus x \oplus 1$, $x^{22} \oplus x \oplus 1$, $x^{23} \oplus x^{15} \oplus x^2 \oplus x \oplus 1$, and $x^{17} \oplus x^5 \oplus 1$, respectively. For the representation of the registers we adopt the notation of [2, 4, 5, 17], in which the bits in the register are given in reverse order, i.e., x^i corresponds to a tap with index $len - i - 1$, where len is the register size. For example, when R4 is clocked, the XOR of R4[$17 - 0 - 1 = 16$] and R4[$17 - 5 - 1 = 11$] is computed. Then, the register is shifted by one bit to the right, and the value of the result of the XOR is placed in R4[0].

At each step of A5/2 R1, R2 and R3 are clocked according to a clocking mechanism that we describe later. Then, R4 is clocked. After the clocking is performed, one output bit is ready at the output of A5/2. The output bit is a non-linear function of the internal state of R1, R2, and R3.

After the initialization 99 bits[5] of output are discarded, and the following 228 bits of output are used as the output key-stream.

Denote the i'th bit of the 64-bit session-key K_c by $K_c[i]$, the i'th bit of register j by $Rj[i]$, and the i'th bit of the 22-bit publicly known frame number by $f[i]$.

The initialization of the internal state with K_c and the frame number is done in the following way:

– Set all LFSRs to 0 ($R1 = R2 = R3 = R4 = 0$).
– For $i := 0$ to 63 do
  1. Clock all 4 LFSRs.
  2. $R1[0] \leftarrow R1[0] \oplus K_c[i]$
  3. $R2[0] \leftarrow R2[0] \oplus K_c[i]$
  4. $R3[0] \leftarrow R3[0] \oplus K_c[i]$
  5. $R4[0] \leftarrow R4[0] \oplus K_c[i]$
– For $i := 0$ to 21 do
  1. Clock all 4 LFSRs.
  2. $R1[0] \leftarrow R1[0] \oplus f[i]$
  3. $R2[0] \leftarrow R2[0] \oplus f[i]$
  4. $R3[0] \leftarrow R3[0] \oplus f[i]$
  5. $R4[0] \leftarrow R4[0] \oplus f[i]$

The key-stream generation is as follows:

[5] 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.

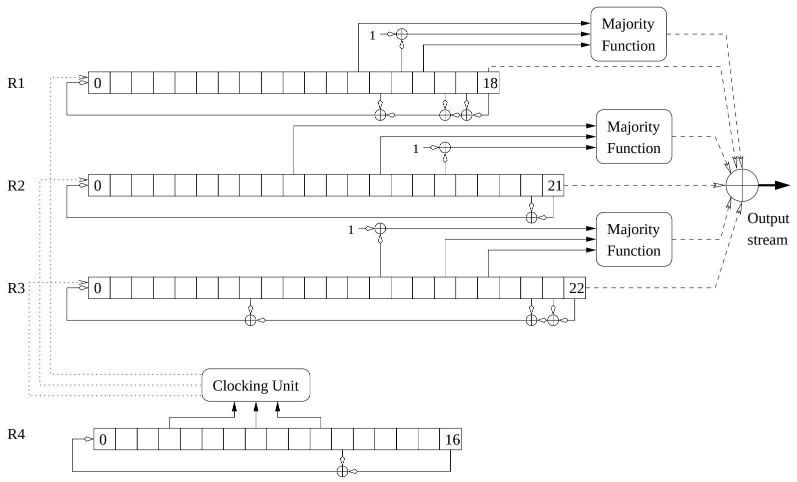

Fig. 1. The A5/2 internal structure

1. Initialize the internal state with K_c and frame number.
2. Force the bits R1[15], R2[16], R3[18], R4[10] to be 1.
3. Run A5/2 for 99 clocks and ignore the output.
4. Run A5/2 for 228 clocks and use the output as key-stream.

After the first clocking is performed the first output bit is ready at the output of A5/2. In Figure 1 we show the internal structure of A5/2. The clocking mechanism works as follows: R4 controls the clocking of R1, R2, and R3. When clocking of R1, R2, and R3 is to be performed, bits R4[3], R4[7], and R4[10] are the input of the clocking unit. The clocking unit performs a majority function on the bits. 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.

Once the clocking is performed, an output bit is ready. The output bit is computed as follows: in each register the majority of two bits and the complement of a third bit is computed; the results of all the majorities and the rightmost bit from each register are XORed to form the output (see Figure 1). Note that the majority function is quadratic in its input: $maj(a, b, c) = a \cdot b \oplus b \cdot c \oplus c \cdot a$.

A5/2 is built on a somewhat similar framework of A5/1. 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 somewhat similar to that of A5/1. The difference is that A5/2 also initializes R4, and that 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 [16].

This algorithm outputs 228 bits of key-stream. The first block of 114 bits is used as a key-stream to encrypt the link from the network to the customer, and the second block of 114 bits is used to encrypt the link from the customer to the network. Encryption is performed as a simple XOR of the message with the key-stream.

Although A5 is a stream cipher, it is used to encrypt 114-bit "blocks", called *frames*. The frames are sequentially numbered (modulo 2^{22}) by a *TDMA frame number*. The frame number f that is used in the initialization of a A5 frame is actually a fixed bit permutation of the TDMA frame number. In the rest of this paper we ignore the existence of this permutation, since it does not affect our analysis.

2.1 GSM Security Background: A3/A8 and GPRS

In this section we give a more detailed description on the usage and specification of A3 and A8. This and more can be found in [12].

A3 provides authentication of the mobile phone to the network, and A8 is used for session-key agreement. The security of these algorithms is based on a user-specific secret key K_i that is common to the mobile phone and the network. The GSM specifications do not specify the length of K_i, thus it is left for the choice of the operator, but usually it is a 128-bit key. Authentication of the customers to the network is performed using the A3 authentication algorithm as follows: The network challenges the customer with a 128-bit randomly chosen value $RAND$. The customer computes a 32-bit long response $SRES = A3(K_i, RAND)$, and sends $SRES$ to the network, which can then check its validity.

The session key K_c is obtained using A8 as follows: $K_c = A8(K_i, RAND)$. Note that A8 and A3 are always invoked together and with the same parameters. In most implementations, they are one algorithm with two outputs, $SRES$ and K_c. Therefore, they are usually referred to as A3/A8.

The security in GPRS is based on the same mechanisms as of GSM. However, GPRS uses a different encryption key to encrypt its traffic. To authenticate a customer, the same A3/A8 algorithm is used with the same K_i, but with a different $RAND$. The resulting K_c is used to encrypt the GPRS traffic. We refer to this key as $GPRS\text{-}K_c$, to differentiate it from K_c which is used to encrypt the GSM voice-traffic. Similarly we refer to the $SRES$ and $RAND$ as $GPRS\text{-}SRES$ and $GPRS\text{-}RAND$ to differentiate them from their GSM-voice counterparts. The GPRS cipher is referred to as GPRS-A5, or GPRS Encryption Algorithm (GEA). There are currently three versions of the algorithm: GEA1, GEA2, and GEA3 (which is actually A5/3).

3 A Known Plaintext Attack on A5/2

In this section we present a new known plaintext attack (known key-stream attack) on A5/2. Namely, given a key-stream, divided to frames, and the respective frame numbers, the attack recovers the session key.

Goldberg, Wagner and Green presented the first attack [13] on A5/2. The time complexity of this attack is very low. However, it requires the knowledge of the XOR of plaintexts in two frames that are 2^{11} frames apart. Their attack shows that the cipher is quite weak, yet it might prove difficult to implement such an attack in practice. The problem is knowing the exact XOR of plaintexts in two frames that are 6 seconds apart. Another aspect is the elapsed time from the beginning of the attack to its completion. Their attack takes at least 6 seconds, because it takes 6 seconds to complete the reception of the data. Our attack might look as if it requires more information, however, it works with only a few milliseconds of data. We improve our attack in Section 4 to a ciphertext-only attack that requires only a few dozen milliseconds of encrypted unknown data. Therefore, our attack is very easy to implement in practice. We have simulated our known plaintext attack on a personal computer, and verified the results. This simulation recovers the key in less than a second. The computation time and memory complexity of this attack are similar to Goldberg, Wagner and Green's attack. The known plaintext attack of Petrović and Fúster-Sabater [17] has similar data requirements as our attack, however, it does not recover the initial key.

Knowing the initial internal state of R1, R2, R3, R4, and the initial frame number, the session key can be retrieved using simple algebraic operations. This is mainly because the initialization process is linear in the session key and the initial frame number. Therefore, in the attack we focus on revealing the initial internal state of the registers.

Let $k_f, k_{f+1}, k_{f+2}, \ldots$ be the output of A5/2 divided to frames. Note that each k_j is the output key-stream for a whole frame, i.e., each k_j is 114-bit long.[6] Let $f, f+1, f+2, \ldots$ be the frame numbers associated with these frames. We denote the i'th bit of the key-stream at frame j by $k_j[i]$. The initial internal state of register Ri at frame j (after the initialization but before the 99 clockings) is denoted by Ri_j.

Assume we know the initial state $R4_f$ of R4 at the first frame. An important observation is that R4 controls the clockings of the other registers. Since we know $R4_f$, then for each output bit we know the exact number of times that a register is clocked to produce that output bit. Each register has a linear feedback, therefore, once given the number of times a register is clocked, we can express every bit of its internal state as a linear combination of bits of the original internal state.

The output of A5/2 is an XOR of the last bits of R1, R2, and R3, and three majority functions of bits of R1, R2, R3 (see Figure 1 for the exact details). Therefore, the resulting function is quadratic with variables which are the bits in the initial state of these registers. We take advantage of this low algebraic degree of the output. The goal in the next paragraphs is to express every bit of the whole output of the cipher (consisting of several frames) as a quadratic

[6] Note that this notation is somewhat imprecise, since the output is actually 228 bits, where the first half is used to encrypt the network-to-mobile link, and the second half is used to encrypt the mobile-to-network link.

multivariate function in the initial state. Then, we construct an overdefined system of quadratic equations which expresses the key-stream generation process and finally we solve it.

Given a frame number f, there is an algebraic description of every output bit. We perform linearization to the quadratic terms in this algebraic description. We observe that each majority function operates on bits of a single register. Therefore, we have quadratic terms consisting 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 $\frac{17*18}{2} = 153$ products. In the same way R2 contributes $21 + \frac{21*20}{2} = 21 + 210$ variables and R3 contributes $22 + \frac{22*21}{2} = 22 + 231$ variables. So far we have $18 + 153 + 21 + 210 + 22 + 231 = 655$ variables after linearization. Together with constant 1 we have a set of 656 variables. We denote the set of these 656 variables by V_f. Of these variables, $18 + 21 + 22 = 61$ variables form the full initial state of R1, R2, and R3.

Every output bit we have adds a linear equation with variables from V_f. A frame consists of 114 bits. Therefore, we get 114 equations from each frame. The solution of the equation system reveals the value of the variables in V_f, and among them the linear variables that directly describe the initial internal state of R1, R2, and R3. However, we do not have enough equations at this stage to efficiently solve the system.

The main observation is that given the variables in V_f defined on frame f, we can describe the bits of any other frame in linear terms of the variables in the set V_f. When moving to the next frame, the frame number is incremented by 1 and the internal state is re-initialized. We assume we know the value of $R4_f$. Due to the initialization method, where the frame number is XORed bit by bit into the registers (see Section 2), we know the value of $R4_{f+1}$. Since we do not know $R1_f$, $R2_f$, and $R3_f$, we do not know the value of $R1_{f+1}$, $R2_{f+1}$, and $R3_{f+1}$, but we know the XOR-differences between $R1_f, R2_f, R3_f$ and $R1_{f+1}, R2_{f+1}, R3_{f+1}$ respectively. We define the set of variables that describe their state and the linearization of these variables as V_{f+1}, in the same way as we did with the first frame to create the set V_f. Due to the initialization method, for each register Ri we know the difference between Ri_{f+1} and Ri_f. Thus, we can describe the variables in the set V_{f+1} as linear combinations of the variables from V_f. We emphasize that even the quadratic terms can be represented in this way. To see this, we assume that $a_{f+1} \cdot b_{f+1}$ is a quadratic term in V_{f+1}, and $a_f \cdot b_f$ is a quadratic term in V_f. We know the differences $d_a = a_{f+1} \oplus a_f$ and $d_b = b_{f+1} \oplus b_f$. Therefore, $a_{f+1} \cdot b_{f+1} = (a_f \oplus d_a) \cdot (b_f \oplus d_b) = a_f \cdot b_f \oplus a_f \cdot d_b \oplus b_f \cdot d_a \oplus d_a \cdot d_b$. Since d_b and d_a are known constants, this equation is linear in the variables in V_f. This fact allows us to use output bits of the second frame in order to get additional linear equations in the variables of V_f. In a similar way, we describe the variables in any set V_i as linear combinations of the variables from V_f. In total, we get an equation system of the form: $S \cdot V_f = k$, where S is the system's matrix, and k is the concatenation of k_f, k_{f+1}, etc.

It is clear that once we obtain 656 linearly independent equations the system can be easily solved using Gauss elimination. However, it is practically very difficult to collect 656 linearly independent equations. This is an effect of the frequent re-initializations, and the low order of the majority function. We note that we do not actually need to solve all the variables, i.e., it suffices to solve the linear variables of the system, since the other variables are defined as their products. We have tested experimentally and found that after we sequentially obtain about 450 equations, the original linear variables in V_f can be solved using Gauss elimination.[7]

We can summarize this attack as follows: we try all the 2^{16} possible values for $R4_f$, and for each such value we solve the linearized system of equations that describe the output. The solution of the equations gives us a suggestion for the internal state of R1, R2, and R3. Together with R4, we have a suggestion for the full internal state. Most of the $2^{16} - 1$ wrong states are identified due to inconsistencies in the Gauss elimination. If more than one consistent internal state remains, these suggestions are verified by trial encryptions.

The time complexity of the attack is as follows: There are 2^{16} possible guesses of the value of $R4_f$. We should multiply this figure by the time it takes to 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 our Linux 800MHz PIII 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 $\approx 54KB$).

3.1 Optimization of the Known Plaintext Attack on A5/2

In an optimized implementation that will be described in detail in the full version of this paper, the average time complexity can be further reduced to about 2^{28} bit-XOR operations (less than 1 second on our personal computer). The memory complexity rises to about $2^{27.8}$ bytes (less than 250MBs). The optimization requires a pre-computation step whose time complexity is about 2^{46} bit-XOR operations (about 160 minutes on our personal computer). The data complexity is slightly higher, and still in the range of a few dozen milliseconds of data. The optimization is based on the observation that for every candidate for the value of $R4_f$ the system of equations contains linearly dependent rows. In the pre-computation stage we consider all the possible values of $R4_f$, for each such value we compute the system of equations. We find which rows in the equation system are linearly dependent by performing a Gauss elimination. In the real-time phase of the attack we filter wrong values for $R4_f$ by checking if the linear dependencies that we found in the pre-computation step hold on the key-stream

[7] 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 [9] can also be used in cases of scarce available data.

bits. This kind of filtering requires two dot products on average for each wrong value of $R4_f$. Once we have a suggestion for the correct value of $R4_f$, the correct key is found using the methods of Section 3.

Note that when using this optimized attack some compromise is needed. Since four known plaintext frames are required, we must know the XOR-differences: $f \oplus (f+1)$, $f \oplus (f+2)$, $f \oplus (f+3)$ in advance, before we know the exact value f. These XOR-differences are required in order to express the frames' key-stream bits as linear terms over the set V_f, and to compute the system of equations. The problematic element is the addition operation, for example, $f+1$ can result in a carry that would propagate through f, thus not allowing the calculation of the XOR-difference in advance. Therefore, we require that f has a specific bit set to 0. This requirement prevents a carry from propagating beyond the specific bit. We require the two last bits in f have a fixed value, and perform the pre-computation for each of the four combinations of the last two bits in f. This requirement is the source of the data complexity being slightly higher, and it also causes a factor four increase in the memory complexity and the pre-computation time complexity.

4 An Instant Ciphertext-Only Attack on A5/2

In this section we convert the attack of Section 3 on A5/2 to a ciphertext-only attack. We observe that error-correction codes are employed in GSM before encryption. Thus, the plaintext has a highly structured redundancy.

There are several kinds of error-correction methods that are used in GSM, and different error-correction schemes are used for different data channels. We focus on control channels, and specifically on the error-correction codes of the Slow Associated Control Channel (SACCH). We note that this error-correction code is the one used during the initialization of a conversation. Therefore, it suffices to focus on this code. Using this error-correction code we mount a ciphertext-only attack that recovers the key. However, we stress that the ideas of our attack can be applied to other error-correction codes as well.

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

The coding operation and the interleaving operation can be modeled together as one 456×184 matrix over $GF(2)$, which we denote by G. The message to be coded is regarded as a 184-bit binary vector, P. The result of the coding-interleaving operation is: $M = G \cdot P$. The resulting vector M consists of 4 frames. In the encryption process each frame is XORed with the output key-stream 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 (and the dimension of the kernel is not larger than 272 due to the properties of the matrix G). In other words, for any vector M, $M = G \cdot P$, there are 272 linearly independent equations on its elements. Let K_G be a matrix that describes these 272 linear equations, i.e., $K_G \cdot M = 0$ for any such M.

We denote the output sequence bits of A5/2 for a duration of 4 frames by $k = k_j || k_{j+1} || k_{j+2} || k_{j+3}$, where $||$ is the concatenation operator. The ciphertext C is computed by $C = M \oplus k$. We use the same 272 equations on C, namely:

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

Since the ciphertext C is known, we actually get linear equations over elements of k. Note that the equations we get are independent of P — they depend only on k. We substitute each bit in k with its description as linear terms over V_f (see Section 3), and thus get equations on variables of V_f. Each 456-bit coding block, provides 272 equations. The rest of the details of the attack and its time complexity are similar to the case in the previous section, but in this attack we know linear combinations of the key-stream, and therefore, the corresponding equations are the respective linear combinations of the equations: Let $S \cdot V_f = k$ be the system of equations from Section 3, where S is the system's matrix. In the ciphertext-only attack we multiply this system by K_G as follows: $(K_G \cdot S) \cdot V_f = (K_G \cdot k)$. K_G is a fixed known matrix that depends only on the coding-interleaving matrix G, S is the system's matrix which is different for each value of $R4_f$ (and for different XOR-differences of $f \oplus (f+1)$, $f \oplus (f+2)$, $f \oplus (f+3)$, etc.). Thus, in the optimized attack, all the possible matrices $K_G \cdot S$ are computed during the pre-computation, and for each such matrix we find linear dependencies of rows by a Gauss elimination. In the real-time phase of the attack we filter wrong values of $R4_f$ by checking if the linear dependencies that we found in the pre-computation step hold on the bits of $K_G \cdot k$.

Note that while four frames of data suffice to launch the attack in Section 3, in the ciphertext-only attack we need eight frames, since from each encrypted frame we get only about half the information compared to the known plaintext attack. The time complexity of the attack is the same as the attack in Section 3.1.

We now analyze the time and memory complexity of this ciphertext-only attack while using the optimized attack of Section 3.1. In Section 3.1 we restrict the values of the three least significant bits of the frame number f, since we need four frames of data. The attack in this section requires eight frames of data, therefore, we restrict the four least significant bits of the frame number f. This restriction doubles the memory complexity compared to the optimized attack of Section 3.1, and it also doubles the pre-computation complexity.

We summarize the complexity of the ciphertext-only attack (using the optimized implementation) as follows: the average time complexity of this ciphertext-only attack is approximately 2^{16} dot products, the memory complexity is about $2^{28.8}$ bytes (less than 500MBs), and the pre-computation time complexity is about 2^{47} bit-XORs. Our implementation on a personal computer (taking advantage of our machine's 32-bit XOR) recovers K_c in less than a second, and it takes about 320 minutes (less than 5.5 hours) for the one-time pre-computation.

We also successfully enhance the attack of Goldberg, Wagner, and Green and the attack of Petrović and Fúster-Sabater to a ciphertext-only attack using our methods. The details of these attacks will appear in the full version of this paper.

5 Leveraging the Attacks to Any GSM Network

The attack shown in Section 4 assumes that the encryption algorithm is A5/2. Using the attack it is easy to recover K_c in real-time from a few dozen milliseconds of ciphertext. We ask the question what happens when the encryption algorithm is not A5/2, but rather is A5/1 or the newly chosen A5/3. The surprising answer is that using weakness of the protocols almost the same attack applies. Also, a variant of the attack works on GPRS networks. All that is needed for the new attack to succeed is that the mobile phone supports A5/2 for voice conversations. The vast majority of handsets support A5/2 encryption (to allow encryption while roaming to networks that use only A5/2).

The following three attacks retrieve the encryption key that the network uses when A5/1 or A5/3 is employed. In the first attack the key is discovered by a man-in-the-middle attack on the victim customer. In this attack, the attacker plays two roles. He impersonates the network to the customer and impersonates the customer to the network. In the second attack, the attacker needs to change bits (flip bits) in the conversation of the mobile and the network (this attack can also be performed as a man-in-the-middle attack). In the third one the attacker impersonates the network for a short radio-session with the mobile. We note that these kind of attacks are relatively very easy to mount in a cellular environment.

This man-in-the-middle attack is performed as follows: when authentication is performed (in the initialization of a conversation), the network sends an authentication request to the attacker, and the attacker sends it to the victim. The victim computes $SRES$, and returns it to the attacker, which sends it back to the network. Now the attacker is "authenticated" to the network. Next, the network asks the customer to start encrypting with A5/1 or A5/3. In our attack, since the attacker impersonates the customer, the network actually asks the attacker to start encrypting with A5/1 or A5/3. The attacker does not have the key, yet, and therefore, is not able to start the encryption. The attacker needs the key before he is asked to use it. To achieve it, the attacker asks the victim to encrypt with A5/2 just after the victim returned the $SRES$, and before the attacker returns the authentication information to the network. This request sounds to the victim as a legitimate request, since the victim sees the attacker as the network. Then, the attacker employs cryptanalysis of A5/2 to retrieve the encryption key of the A5/2 that is used by the victim. Only then, the attacker sends the authentication information to the network. The key only depends on $RAND$, that means that the key recovered through the A5/2 attack is the same key to be used when A5/1 is used or even when 64-bit A5/3 is used! Now the attacker can encrypt/decrypt with A5/1 or A5/3 using this key.

Some readers may suspect that the network may identify this attack, by identifying a small delay in the time it takes to the authentication procedure to complete. 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.

A second possible attack, which can be relatively easily spotted (and prevented) by the network, is a class-mark attack. During initialization of conversation, the mobile phone sends his ciphering capabilities to the network (this information is called class-mark). Most mobile phones currently support A5/1, A5/2, and A5/0 (no encryption), but this may change from phone to phone, and can change in the future. The attacker changes (for example using a man-in-the-middle attack) the class-mark information that the mobile phone sends, in a way that the network thinks that the mobile phone can only support A5/2, and A5/0. The network then defaults to A5/2, and thus allowing the attacker to listen to the conversation. This attack takes advantage of the following protocol flaw: the class-mark information is not protected.

Many networks initiate the authentication procedure rarely, and use the key created in the last authentication. An attacker can discover this key by impersonating the network to the victim mobile phone. Then the attacker initiates a radio-session with the victim, and asks the victim mobile phone to start encrypting using A5/2. The attacker performs the attack, recovers the key, and ends the radio session. The owner of the mobile phone and the network have no indication of the attack.

The leveraging in the first and last attacks relies on the fact that the same key is loaded to A5/2 and A5/1 and even to 64-bit A5/3 (in case A5/3 is used in GSM, according to GSM standards). Thus, discovering the key for A5/2 reveals the key for A5/1 and 64-bit A5/3. We note that although A5/3 can be used with key lengths of 64-128 bits, the GSM standard allows the use of only 64-bit A5/3.

A similar attack can be performed on GPRS. The attacker listens to the $GPRS\text{-}RAND$ sent by the network to the customer. The attacker can impersonate the voice network, initiate radio session with the customer and start authentication procedure using the $GPRS\text{-}RAND$ value that he intercepted, as the GSM-voice $RAND$. The result is that K_c equals $GPRS\text{-}K_c$. The attacker asks the customer to encrypt with A5/2, recovers K_c, and ends the radio-session. 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 in order to retrieve the $GPRS\text{-}K_c$ with which the recorded data can be decrypted. If A5/2 is not supported by the phone, but A5/1 is supported, then the above attacks against A5/3 and GPRS can be performed using the (more complex) cryptanalysis of A5/1 instead of A5/2 as a building block.

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

In this section we discuss possible ciphertext-only attacks on GSM communications that are encrypted using A5/1. Unlike Section 5, this section discusses passive attacks, i.e., these attacks do not transmit any radio signals. Our starting point for the passive ciphertext-only attacks is that error-correction-codes are employed before encryption, as discussed in Section 4.

Available data D	M	Number of 200GBs disk	P=N/D	Number of PCs to complete preprocessing in one year	T	Number of PCs to complete attack in real-time
2^{12} (≈ 5 min)	2^{38}	≈ 22	2^{52}	140	2^{28}	1
$2^{6.7}$ (≈ 8 sec)	2^{41}	≈ 176	$2^{57.3}$	5000	$2^{32.6}$	1000
$2^{6.7}$ (≈ 8 sec)	2^{42}	≈ 350	$2^{57.3}$	5000	$2^{30.6}$	200
2^{14} (≈ 20 min)	2^{35}	≈ 3	2^{50}	35	2^{30}	1

Table 1. Four points on the time/memory/data tradeoff curve

We apply a time/memory/data tradeoff, similar to the one presented by Biryukov, Shamir and Wagner [4], and further discussed and generalized by Biryukov and Shamir [3]. Their original attack requires about two second of known plaintext, and takes a few minutes to execute on a personal computer. It also requires a preprocessing time of about 2^{48} and memory of four 73GByte disks.

In our attack, we are given a block of four encrypted frames. We use the methods of Section 4 to compute $K_G \cdot k$, which is 272 bits long. These 272 bits depend only on the output of A5/1 for four consecutive frames. We call these output bits the *coded-stream*. Let's assume that we know that the frame number of the first frame of these four frames is divisible by four without remainder — this assumption limits us to use only a quarter of the GSM data stream on average. We can view the whole process as a function from the internal state of A5/1 to the coded-stream. Let $h : \{0,1\}^{64} \rightarrow \{0,1\}^{272}$ be the function that takes an internal state of A5/1 after initialization, and outputs the coded-stream. Inverting h reveals the internal state, and breaks the cipher. Note that the output of h is calculated from the key-stream of four frames, while its input is the internal state after the initialization of A5/1 at the first frame. The assumption we make on the frame numbers facilitates the computation of the initial internal states at the other three frames from the initial internal state at the first frame. It suffices to look at only 64 bits of the output of h, therefore, we can assume that $h : \{0,1\}^{64} \rightarrow \{0,1\}^{64}$. We apply Biryukov and Shamir's time/memory/data tradeoff attack with sampling [3] and adopt their notations: the number of possible states is $N = 2^{64}$, the number of four-frame blocks that we allow is denoted by D, T is the number of applications of h during the real-time phase of the attack, M is the number of memory rows (in our case each row is about 16-byte long), and the tradeoff curve is: $TM^2D^2 = N^2$, $D^2 \leq T \leq N$. The preprocessing complexity $P = N/D$ is the number of applications of h during preprocessing. The attack performs about $\sqrt{T}$ memory accesses. We assume that h can be applied 2^{20} times every second on a personal computer. We list four points on the tradeoff curve in Table 1.

A more detailed description about the passive attack will appear in the full version of this paper.

7 Possible Attack Scenarios

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

7.1 Call Wire-Tapping

The most naive scenario that one might anticipate is eavesdropping conversations. Communications encrypted using GSM can be decrypted and eavesdropped by an attacker, once the attacker has the encryption key. Both voice conversations and data, for example SMS messages, can be wire-tapped. Video and picture messages that are sent over GPRS can also be tapped, etc. These attacks are described in Section 5.

In another possible wire-tapping attack the attacker records the encrypted conversation. The attacker must make sure that he knows the $RAND$ value that created the used key (the $RAND$ is sent unencrypted). At a later time, when it is convenient for the attacker, the attacker impersonates the network to the victim. Then the attacker initiates a GSM radio-session, asks the victim to perform authentication with the above $RAND$, and recovers the session key used in the recorded conversation. Once the attacker has the key he simply decrypts the conversation and can listen to its contents. 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.

7.2 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 do not have K_c, and thus cannot conduct encrypted communications. We show how to obtain encryption keys. Once an attacker has the encryption keys, he can cut the victim off the conversation, and impersonate the victim to the other party. 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 there is an attack. The only clue of an attack is a moment of some increased electro-magnetic interference.

Another way to hijack incoming calls is to mount a kind of a man-in-the-middle attack, but instead of forwarding the call to the victim, the attacker receives the call.

7.3 Altering of Data Messages (SMS)

Once a call has been hijacked, the attacker decides on the content. The attacker can listen to the contents of a message being sent by the victim, and send his own version. The attacker can stop the message, or send his own SMS message. This compromises the integrity of GSM traffic.

7.4 Call Theft — Dynamic Cloning

GSM was believed to be secure against call theft, due to authentication procedures of A3/A8 (at least for operators that use a strong primitive for A3/A8 rather then COMP128).

However, due to the mentioned weaknesses, an attacker can make outgoing calls on the expense of a victim. When the network asks for authentication, a man-in-the-middle attack similar to the one described in Section 5 can be applied: 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 can also recover K_c as described in Section 5. 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. At least until his monthly bill arrives.

8 Summary

In this paper we present new methods for attacking GSM and GPRS encryption and security protocols. 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 early as possible, or switch to the more secure third generation cellular system.

In GSM, even GSM networks using the new A5/3 succumb to our attack. We suggest to change the way A5/3 is integrated 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 system's security.

Acknowledgments

We are grateful to Orr Dunkelman for his great help and various comments on early versions of this paper, 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.

References

1. *The 3rd Generation Partnership Project (3GPP)*, `http://www.3gpp.org/`.
2. 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.
3. Alex Biryukov, Adi Shamir, *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*, Advances in Cryptology, proceedings of Asiacrypt'00, Lecture Notes in Computer Science 1976, Springer-Verlag, pp. 1–13, 2000.
4. 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.
5. 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.
6. Marc Briceno, Ian Goldberg, David Wagner, *An implementation of the GSM A3A8 algorithm*, `http://www.iol.ie/~kooltek/a3a8.txt`, 1998.
7. Marc Briceno, Ian Goldberg, David Wagner, *GSM Cloning*, `http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html`, 1998.
8. Nicolas Courtois, *Higher Order Correlation Attacks,XL Algorithm and Cryptanalysis of Toyocrypt*, proceedings of ICISC'02, Lecture Notes in Computer Science 2587, Springer-Verlag, pp. 182–199, 2003.
9. Nicolas Courtois, Alexander Klimov, Jacques Patarin, Adi Shamir, *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*, Advances in Cryptology, proceedings of Eurocrypt'00, Lecture Notes in Computer Science 1807, Springer-Verlag, pp. 392–407, 2000.
10. Nicolas Courtois, Josef Pieprzyk, *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations*, Advances in Cryptology, proceedings of Asiacrypt'02, Lecture Notes in Computer Science 2501, Springer-Verlag, pp. 267–287, 2002.
11. Patrik Ekdahl, Thomas Johansson, *Another Attack on A5/1*, to be published in IEEE Transactions on Information Theory, `http://www.it.lth.se/patrik/publications.html`, 2002.
12. 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`.
13. Ian Goldberg, David Wagner, Lucky Green, *The (Real-Time) Cryptanalysis of A5/2*, presented at the Rump Session of Crypto'99, 1999.
14. Jovan Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, Advances in Cryptology, proceedings of Eurocrypt'97, LNCS 1233, pp.239–255, Springer-Verlag,1997.
15. Aviad Kipnis, Adi Shamir, *Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization*, Advances in Cryptology, proceedings of Crypto'99, Lecture Notes in Computer Science 1666, Springer-Verlag, pp. 19–30, 1999.
16. 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.
17. Slobodan Petrović, Amparo Fúster-Sabater, *Cryptanalysis of the A5/2 Algorithm*, Cryptology ePrint Archive, Report 2000/052, `http://eprint.iacr.org`, 2000.