A Hardware-Based Attack on the A5/1 Stream Cipher

Prof. Dr. Jörg Keller, FernUniversität, 58084 Hagen, Germany

Dipl.-Inf. Birgit Seitz, Rohde & Schwarz GmbH & Co. KG, 81614 München, Germany

Abstract

We present a known-plaintext attack on the A5/1 stream cipher, the encryption algorithm used by the GSM customers in Europe during their conversations with cellular phones. The attack differs from previous approaches in two aspects: it only needs a very small amount of plaintext, and it is not solely based on software. A crucial part of the attack algorithm is implemented in a field programmable gate array (FPGA). We present performance figures which suggest that a distributed implementation on 1,000 ASICs could recover a session key in less than a minute, from which on the conversation could be deciphered by an eavesdropper in real-time. We conclude that, at least for longer conversations, A5/1 is not secure and that its replacement might be even more urgent than for the DES algorithm, where a successor already has been announced.

1 Introduction

The A5/1 algorithm is the encryption algorithm used in the GSM system, i.e. during calls with cellular phones in Europe. With such a wide deployment, its security against attacks is a crucial issue. However, several known-plaintext attacks on the A5/1 have already appeared in the literature, see [1] for a survey. A common feature of these attacks is that they are performed in software alone and need several minutes to derive the key, and thus are not really to be considered real-time attacks. Also, they need at least several seconds of plaintext data to start work.

As known from the DES cracker [2], the implementation of an attack in simple hardware units can make even simple approaches feasible at a — considering the potential of such a machine — moderate cost.

We present an attack on the A5/1 cipher that draws ideas from previous attacks but needs only a few frames of plaintext. For the time-consuming part of the attack, we provide an implementation for a field programmable gate array (FPGA), and give performance estimations for this implementation. We discuss a possible, distributed re-implementation for application specific integrated circuits (ASIC). Our conclusion is that this re-implementation could yield a performance increase by a factor such that the computation of a session key would be possible in less than a minute on a machine containing about 1,000 cracker ASICs.

The remainder of the paper is organized as follows. In Section 2 we briefly review the A5/1 algorithm. In Section 3 we review previous attacks on the A5/1 and present our attack. In Section 4 we discuss the performance of an FPGA implementation and the projected performance in a distributed implementation on ASICs. In Section 5, we summarize and conclude.

2 The A5/1 Algorithm

In conversations according to the GSM standard, the communicating parties exchange data in frames of length 228 bits (114 bits in each direction) every 4.6 milliseconds. For the transmission between the cellular phone and the base station, the data is encrypted.

Each frame is encrypted by a bitwise exclusive-or (addition modulo 2 in GF(2)) with the output of the stream cipher generator A5/1; decryption happens in the same manner, see **Figure 1**. Before encrypting or decrypting a frame, the generators on both sides of a communication are initialized with a key that is fixed during the call, and a frame number which is publicly known [1].

The A5/1 algorithm was not made public by the GSM organization. The following description is based on [1], which in turn refers to a description at `http://www.scard.org` based on reverse engineering of a GSM telephone; the latter description is referenced as being confirmed to be correct by the GSM organization.

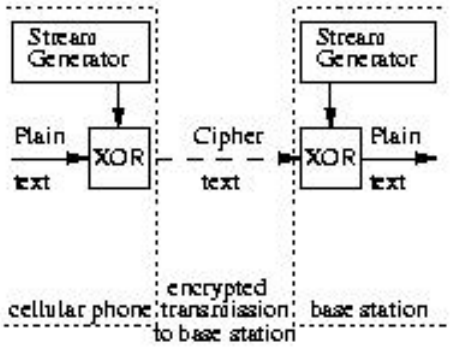

Figure 1: Use of the A5/1 generator in GSM.

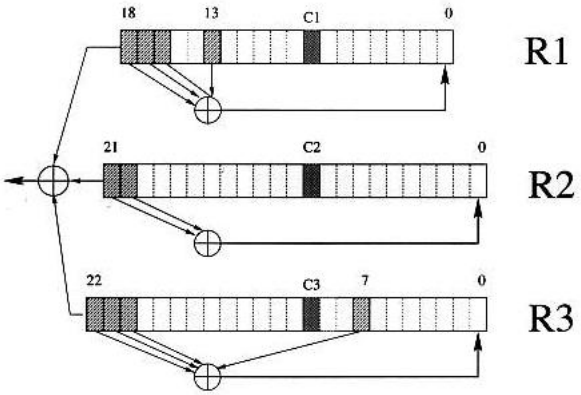

Figure 2: Construction of the A5/1 generator in GSM.

The A5/1 generator consists of three linear feedback shift registers (LFSRs) called R1, R2, R3 of lengths 19, 22, and 23 bits respectively, see **Figure 2**. The most significant bits of the registers are processed by an exlusive-or operation, which forms the output of the generator. The feedback for each register, i.e. the value to be entered at the least significant bit when the register is clocked, is formed by an exclusive-or of several of the register's bits. These so-called tap bits are at positions 13, 16, 17, 18 for R1, 20, 21 for R2, and 7, 20, 21, 22 for R3. The decision whether a register is clocked in normal operation is done by the following "clock control". Bit 8 of R1, and bits 10 of R2 and R3 form the clock bits, of which a majority value is computed. A register is clocked if its clock bit contains a value equal to this majority value. If the clock control is disabled, then each register is clocked.

The generator is initialized for every frame with a 64-bit session key, which is secret and fixed for the conversation, and a publicly known 22-bit frame number. The initialization happens by setting all registers to zero and clocking *all* registers for $64 + 22 = 86$ cycles, using the key bits and the frame number bits as additional feedback bits. Then the generator is run for 100 cycles with the above clock control enabled, the output is not used. Then, the initialization is complete and the sequence O of the next 228 output bits is used to encrypt the part of the frame to be sent and to decrypt the part of the frame which was received.

3 Attacks on the A5/1

All known attacks (see [1] for a survey) are known-plaintext attacks. We too make the assumption that we are able to intercept an encrypted frame and that we are in possession of the corresponding plaintext data. We also make the assumption that we are in possession of the publicly known frame number. While the first assumption is common in the cryptanalysis of ciphers [4] and we do not want to dive into the details of data interception, we would like to give some support why this assumption is feasible. The frames are used to transmit both signalling and voice data. The signalling data have a specific format and hence, if we are able to intercept an encrypted frame with such data, we will with a certain probability know what its content (the plaintext) will be. This is especially true as our attack only needs a small number of frames (normally only one), whereas other attacks assume to be in possession of seconds of plaintext frame data.

The ciphertext was obtained by a bitwise exclusive or of the plaintext and the output O of the generator. Hence, by computing the bitwise exclusive or of the plaintext and the ciphertext, we obtain the output of the stream generator used to encrypt the plaintext. The task of the attack is to find the session key from this output and the frame number. Then, any future frame of this conversation can be decrypted.

Finding the key is equivalent to computing the state of the three registers R1, R2, R3 after the first 64 cycles of the initialization, as this state only depends on the key, and can be used as starting point for the decryption of all future frames during the intercepted conversation.

A simple attack could now be the following: for each of the $2^{19+22+23} = 2^{64}$ possible states of the three registers after input of the key, perform the remaining $22 + 100 = 122$ cycles of the initialization and check whether the known output sequence O is generated. In the worst case, this attack takes $2^{64} \cdot (122 + 228) \approx 1.37 \cdot 2^{72}$ cycles on a device that performs one cycle of the generator at a time.

Instead, we first test for each possible state S after the end of the initialization whether the generator, starting in state S, generates the output sequence O. We will deal with this test in more detail in the remainder of the section, as it constitutes the majority of work.

For each of the few states S that generate O, one computes the corresponding state S'' after the key has been input — if such a state exists[1]— by a 2-phase backtracking algorithm. The first phase tries to generate from S the state S' after the frame number has been input, i.e. 100 cycles before S was reached. As not every register is clocked in each cycle due to the clock control, there could be up to 4 predecessor states for each cycle. However, as the average number of predecessor states is 1,

[1]The state transistion function of the A5/1 algorithm is not injective [3].

the number of possible states in the tree of predecessor states grows only linearly with the height of the tree [3].

The second phase consists of computing, for each state S' found, the corresponding state S'' of the registers after the input of the key, i.e. 22 cycles before S' was reached. As in these cycles all registers are clocked, and the frame number is known, state S'' can be generated easily if it exists.

In [3] it is shown that already for the first 64 bits of the output sequence O, there will be only one state S'' leading to O with very high probability, hence, the backtracking will derive the actual session key. If this is not the case, one has to repeat the procedure with a different frame.

Let us return to the test of all states S that generate the output sequence O. A simple test of all states is no advantage over the simple attack described at the beginning of the section, as 2^{64} states have to be tested, and each test takes 228 cycles in the worst case. Instead, we compute for each of the $2^{19+22} = 2^{41}$ states possible for R1 and R2 at the end of the initialization, all states that R3 could have had at that time, given the state of R1 and R2 and the output sequence O. For this computation, only the first bits of O are inspected. As soon as the state of R3 is available, we test whether the sequence O is indeed generated.

As the first bit of the output sequence O is generated by an exclusive-or of the most significant bits of R1, R2, and R3, we can immediately derive the most significant bit of R3 from the knowledge of R1, R2, and O. We clock the registers, and can again derive the most significant bit of R3 from the most significant bits of R1 and R2 and the second bit of O. The bit now derived had been next to the most significant bit in R3, i.e. in position 21, in the previous cycle. Hence, by proceeding for at most 23 cycles, we can derived all bits of R3.

The difficulty lies in the fact that we do not know which registers will be clocked in each cycle, as the clock control in the first 11 cycles depends on the values of the bits 0 to 10 of register R3 at the end of the initialization. However, we derive these bits only at later stages of the check. In order to avoid going over all 2^{11} possible values of these bits, we proceed as follows.

If the clock bits of R1 and R2 are identical, then both registers are clocked. If we assume the clock bit of R3 to be different, then R3 will not be clocked, and its most significant bit in the next cycle will remain the same. The output bit generated by an exclusive-or operation of the most significant bits can then be compared to the bit of the output sequence O. If they differ, then this possibility was a false one and the clock bits must be equal. If the output bit produced and the bit from the output sequence are identical, we pursue this possibility. Thus, in one half of the situations (R1 and R2 having identical clock bits) on the average, we reduce the number of possibilities from two to one and have to check $(3/2)^{11} \approx 85$ cases. For each possibility, after 11 cycles one can check whether the assumption about the clock bits in previous cycles matches with the generation of the most significant bit 12 cycles later. Then, false possibilities can be eliminated fast.

This heuristic reduces the number of possibilities to be checked to 85 on the average, with 14 output sequence bits inspected on the average [5].

4 A Hardware-Based Attack

We decided to invest only the hardware implementation of the test that we described in detail in the previous section. As only some states S will produce the output sequence O, the generation of corresponding states S'' can be left to software, as the test will dominate cost and runtime of the attack.

We have implemented this test in VHDL and compiled it with the Xilinx Foundation software for an Xilinx XC4062 FPGA [5]. The implementation occupies 313 of its 2300 CLBs. Therefore, 7 instances of the test are possible on one FPGA, each doing its share of the 2^{41} checks. Due to the complexity of the control logic to generate the bits of register R3, the frequency that could be achieved was pretty low: 18.65 MHz.

On the average, 85 possibilities for register R3 have to be checked for each state of R1 and R2; for each of these possibilities, 14 bits of the output sequence have to be checked on the average. Thus, a check needs $85 \cdot 14 = 1190$ cycles or 63.8 μs on the average.

We have done an estimation of the cost and performance of an implementation of the test on an Alcatel ASIC with about 5,000 gate equivalents in 0.35μ technology [5]. Such an ASIC could host 11 instances of the test and would yield a guaranteed operating frequency of at least 50 MHz, i.e. the test on an ASIC would be faster than on an FPGA by a factor

$$\frac{50 \cdot 11}{18.65 \cdot 7} \approx 4.2$$

.

A projection of this implementation onto a 0.10μ technology [5] resulted in a frequency increase by a factor three and an area decrease by a factor 16. Hence, with the same chip area, a soon to be available ASIC could be $16 \cdot 3 \cdot 4.2 \approx 200$ times faster than the FPGA implementation.

Taking into consideration that ASICs not only get faster with shrinking feature sizes but also tend to provide more chip area, we argue that with 1,000 ASICs in 0.1μ technology, each providing 10 times more chip area than the ASIC we considered, we get the following calculation.

One FPGA would need to perform 2^{41} checks, each needing 63.8 μs on the average. A cheap ASIC available today would by 200 times faster. 1,000 larger ASICs then would need

$$\frac{2^{41} \cdot 63.8}{200 \cdot 1000 \cdot 10} \cdot 10^{-6}\text{s} \approx 70\,\text{s}$$

Given the fact that on the average, only half of the checks have to be performed until the correct state is found, a machine containing these 1,000 ASICs would need roughly half a minute to find the key of a phone call from a single frame of the conversation.

Such a machine would be very similar to the DES cracker [2], that hosted 1,536 cracker chips and a normal PC as a host computer. The cracker chips are connected to the host computer by

a hierarchical bus system. In the beginning, they are initialized, and each chip receives the set of states it has to check. The set is typically given as an interval of states. During operation, a chip reports each state found to the host computer. As the number of potential solutions is small, a bus system is sufficient. The cost of the DES cracker was estimated between 1 and 2 million US$. Our machine would be of similar complexity.

5 Conclusions

We have presented a simple-to-implement known-plaintext attack on the A5/1 stream cipher, and given an implementation on a small FPGA. The attack is novel over previous attacks in that it needs only a very small amount of plaintext frame data. A distributed implementation on specialized hardware was projected to derive a key within half a minute on the average. We conclude that the A5/1 algorithm is not secure for longer phone calls.

We have not taken advantage of approaches by others [1] that try to restrict the number of possible states. This can be done either by taking into account that in known implementations, 10 bits of the session keys are always set to zero, or by precomputing large tables of states that are indexed via the first bits of the output sequence [1].

If we combine our effort with theirs, the time to derive a key will shrink or smaller machines to attack A5/1 will suffice, making the attack even more realistic. It might be suspected that such machines are already in use by the National Security Agency (NSA) or other, similar organizations.

References

[1] Alex Birykov, Adi Shamir, David Wagner. Real Time Cryptanalysis of A5/1 on a PC. Presented at the *Fast Software Encryption Workshop, April 10-12, 2000, New York, NY*. Available at `http://cryptome.org/a51-bsw.htm`

[2] Electronic Frontier Foundation (EFF). *Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design*. O'Reilly & Associates, Sebastopol 1998.

[3] Jovan Dj. Golic. Cryptanalysis of Alleged A5 Stream Cipher. In *Proc. Eurocrypt'97*, pp. 239–255, Springer Verlag 1997.

[4] Bruce Schneier. *Applied Cryptography 2nd Edition*. Wiley & Sons, New York 1996.

[5] Birgit Seitz. Krypto-Cracker auf FPGA-Basis. Diplomarbeit, FernUniversität Hagen, FB Informatik, Nov. 2000.