Implementation of a Brute Force Attack on the A5/1 Keystream Generator in a GPU-based Volunteer Computing Project*

Vadim Bulavintsev Alexander Semenov
Oleg Zaikin
Matrosov Institute for System Dynamics and Control Theory SB RAS
Irkutsk, Russia
v.g.bulavintsev@gmail.com, biclop.rambler@yandex.ru, zaikin.icc@gmail.com

Abstract

We present an advanced brute force attack on the A5/1 keystream generator, that is still widely used in modern GSM networks. To greatly reduce the search space, we use a well-known idea, introduced by R. Anderson more than 20 years ago. The main contribution of the present paper is the implementation of Anderson's attack on a GPU platform in bit-slice technique. The preliminary estimates of the attack's speed showed that, with the use of GPUs processing power, the attack could be performed in the real time on a modern computer cluster or in a volunteer computing project. To verify our estimates with the use of the BOINC technology we launched a volunteer computing project and executed our variant of Anderson's attack within it. As a result, 10 A5/1 cryptanalysis problems were solved in 7 days in the project. The results presented in this work provide yet another proof of A5/1's cryptographic weakness that shows that this generator is totally unsuitable for transmission of any kind of sensitive data through modern GSM networks.

1 Introduction

The A5/1 keystream generator has a key length of 64 bits. It is used to encrypt voice and SMS traffic in 2nd generation (2G) GSM networks. A5/1 is one of the most publicly recognized cryptographic algorithms.

The growth of the computational power of GPUs and FPGAs made it possible to put into practice the attack on A5/1, which was described by R. Anderson in 1994 [And]. This attack is based on reduction of the search space size from 2^{64} to 2^{53}. The FPGA-based variant of Anderson's attack was already performed in 2008 in the COPACOBANA project [GNR08]. So, the primary goal of our work is to demonstrate the viability of GPU-based variant of the attack. Let us note that GPUs are much easier to operate than FPGAs. Besides, the former belong to the class of consumer-grade devices and could be found in any modern PC, while the latter belong to the class of specialized equipment. With the usage of the BOINC software platform [And04], these qualities of GPUs allowed us to implement the attack in the form of a volunteer computing project using idle computational

*The research was funded by Russian Science Foundation (project No. 16-11-10046).

In: E. Ivashko, A. Rumyantsev (eds.): Proceedings of the Third International Conference BOINC:FAST 2017, Petrozavodsk, Russia, August 28 - September 01, 2017, published at http://ceur-ws.org

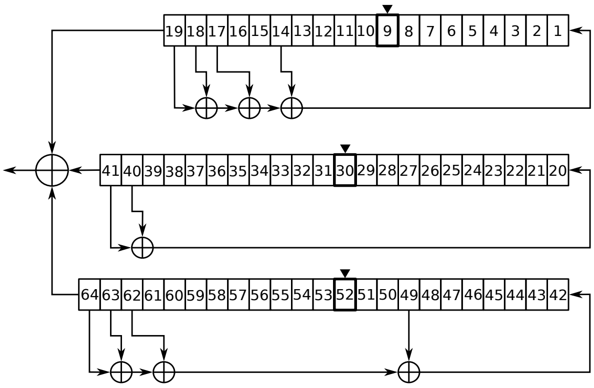

Figure 1: The A5/1 generator scheme

capabilities of the project members home PCs. Our estimates of the attack's speed were based on our previous work[BS16].

Let us make a brief outline of the article's contents. In Section 2 we describe the A5/1 algorithm along with some advanced brute force attacks on it. Section 3 introduces bit-slicing technique and goes through important details of implementing Anderson's attack. Section 4 provides a look into the internal organization of the volunteer project which was launched by us to perform the attack. Section 5 contains the retrospective of A5/1 cryptanalysis works related to our study.

2 A5/1 keystream generator

The A5/1 keystream generator consists of 3 linear feedback shift registers (LFSRs [MVO96]), defined by the following primitive polynomials:

$$LFSR1 : X^{19} + X^{18} + X^{17} + X^{14} + 1;$$
$$LFSR2 : X^{22} + X^{28} + 1;$$
$$LFSR3 : X^{23} + X^{22} + X^{21} + X^{8} + 1.$$

The illustration of the A5/1 generator's scheme can be seen at Figure 1.

The outputs of the LFSRs are mixed by a linear function (addition modulo 2), that provides the best correlation immunity. Non-linearity of the cryptanalysis equations is achieved by clocking the registers asynchronously — on each clocking of the generator any of it's LFSR can be shifted or it can retain its current state. LFSR with index $j \in \{1, 2, 3\}$ is shifted if the following Boolean function χ_j takes the value of 1:

$$\chi_j = (b_j \equiv majority(b_1, b_2, b_3));$$
$$majority(A, B, C) = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C).$$

Here b_1, b_2, b_3 denote clocking bits marked at Figure 1 by black wedges. Conversely, if at some moment $\chi_j = 0$, LFSRj is not shifted (it remains in its last state).

Further, we will focus on the idea of the attack that was suggested by R. Anderson in 1994 in a small essay on the A5/1 cryptographic resistance [And]. Next we describe the details of Anderson's attack.

Anderson's attack is a typical example of a *guess and determine attack* (see, for example, [Bar09]). Suppose that we know the bits filling LFSR1 and LFSR3, and bits of LFSR2 from the beginning of the register to the clocking bit (bits 31 to 41, see Figure 2). Next, suppose that we know 64 bits of the keystream. It was shown by R. Anderson, that 11 unknown bits of LFSR2 can be figured out without any additional guesses. This is possible because the following data is known:

- the clocking bits (so, the clocking schedule for the next 11 shifts of LFSR2 is also known);
- 2 out of 3 LFSRs output bits, which are used as input for the XOR operation;

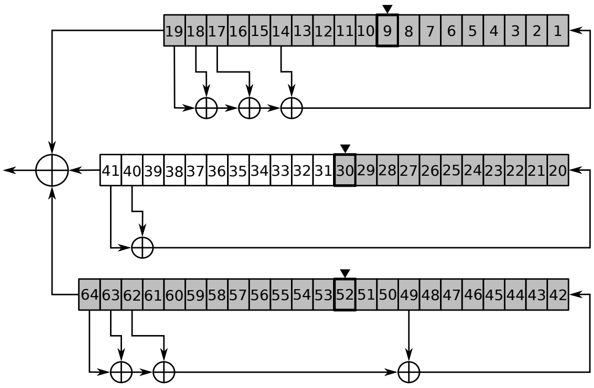

Figure 2: The set of guessing bits used in Anderson's attack (greyed out).

- the result of the XOR operation (from the keystream).

Therefore, one can efficiently derive the unknown bits of LFSR2 one by one, by clocking the generator and applying XOR operation to corresponding keystream bits and output bits of LFSR1 and LFSR3.

The considered algorithm, that is used to determine the unknown 11 bits of LFSR2, makes it possible to mount a brute force attack on the A5/1 generator over the search space with the size of 2^{53}. The simplicity of the algorithm provides an opportunity to implement it on a specialized computational architecture. One such implementation was built with FPGAs by authors of [GNR08]. In the following sections we describe our implementation of this attack for modern GPUs.

3 Bit-slicing-based Implementation of Anderson's Attack

The efficiency of a brute force attack is defined by two parameters: the speed of checking of the key candidates and the size of the search space. R. Anderson's idea described in the previous section gives us the search space with the size of 2^{53}. Instead of using the "naive" implementation of the A5/1 generator, to speed up the key candidates checking procedure one can opt to use more sophisticated alternatives. In [BS16] we evaluated the performance of two different fast implementations of A5/1 generator. The first one was based on an idea of precomputation of the states of LFSR1-3, and keeping these states in PC's memory. This approach demonstrated a considerable speed-up against "naive" implementation. A similar method of precomputation of LFSRs was described in [BSW00]. However, in [BS16] we found its performance inferior to another one implementation, that is based on *bit-slicing technique*. Next we briefly describe the idea of this technique, and its application to Anderson's attack.

Consider an arbitrary total Boolean function $f : \{0,1\}^n \to \{0,1\}$. This function can be represented in the form of the Boolean circuit $C(f)$ over some complete basis B. A common example of such basis is $B = \{\wedge, \vee, \neg\}$, but we will use the basis $B = \{\wedge, \vee, \neg, \oplus\}$ instead, since it better fits our goals.

Now consider the problem of calculating the arbitrary total Boolean function $f : \{0,1\}^n \to \{0,1\}$ over all 2^n possible inputs. For each input $X \in \{0,1\}^n$ one can calculate the value of f as a superposition of the basis functions, according to the circuit $C(f)$. We can select a fixed order of calculation of basis functions from $C(f)$, that results in getting the value of f. Let m be the number of internal nodes in $C(f)$. Assuming that the calculation of one basis function takes one processor instruction, the computation of f over all inputs from $\{0,1\}^n$ will take $m \cdot 2^n$ instructions.

SIMD(Single Instruction, Multiple Data) architecture calculates many copies of the same function over many different memory cells with a single instruction. When a modern computational device executes a bitwise logical instruction on its general-purpose registers (GPRs), it effectively acts as a SIMD device, in which individual bits of GPRs play the role of the individual memory cells. The calculation order of functions in the circuit $C(f)$ always stays the same. This makes it possible to compute this function over as many inputs, as there are bits in the device's GPR. If D is the device's GPR capacity, we can simultaneously process D instances of the circuit $C(f)$, calculating the value of f for inputs $X_1, ..., X_D$.

Consider the arbitrary basis function g with arity 2, and the corresponding internal node of the circuit $C(f)$. We denote it as $G(x_1, x_2)$, meaning that it has a single output and two inputs, which values are determined by Boolean variables x_1, x_2. Next, let us link g with three GPRs denoted $R_1(g), R_2(g), R_3(g)$, each of which is comprised of D single-bit memory cells, filled in the following way:

- register R_1 contains D values of the variable x_1, corresponding to $X_1, ..., X_D$;
- register R_2 contains D values of the variable x_2, corresponding to $X_1, ..., X_D$;
- register R_3 contains D matching values of the function g.

Suppose that all D instances of g can be computed as a result of a single bitwise instruction applied to registers $R_1(g), R_2(g)$, while their result is put into register $R_3(g)$. If this fact holds for every basis function in the circuit $C(f)$, the computation of f for all inputs from $\{0,1\}^n$ will require $m \cdot \frac{2^n}{D}$ instructions. This is the key idea of bit-slicing technique.

We will call the process of computation of the function f (represented by the circuit $C(f)$ with m inner nodes) on a single input from $\{0,1\}^n$ a *thread*, by analogy with the computational threads in a SIMD device. Thus, with the use of bit-slicing technique, it takes m instructions to execute D threads.

Now we describe the details of bit-slicing implementation of the A5/1 generator. Suppose that a computational device is able to calculate D instances of any function from the basis $B = \{\wedge, \vee, \neg, \oplus\}$. Each generator's cell with number $n, n \in \{1, ..., 64\}$ gets a corresponding word $W_n \in \{0,1\}^D$:

$$
\begin{aligned}
LFSR1 &: W_1, ..., W_{19}; \\
LFSR2 &: W_{20}, ..., W_{42}; \\
LFSR3 &: W_{43}, ..., W_{64}.
\end{aligned}
$$

In bit-slicing technique, the shifting of the LFSR register (LFSR1 in this example) will take the following form:

$$
\begin{aligned}
W_1' &= W_{19} \oplus W_{18} \oplus W_{17} \oplus W_{14}, \\
W_n' &= W_{n-1}, n \in \{2, ..., 19\},
\end{aligned}
$$

where $\oplus$ is the bitwise addition modulo 2 of Boolean vectors of length D. The calculation of the keystream bit will look like:

$$W_{out} = W_{19} \oplus W_{41} \oplus W_{64}.$$

The conditional clocking is somewhat more complex to implement in bit-slicing technique. First, to know if the LFSRs should be shifted or not, one needs to calculate the corresponding shifting flags F_1, F_2, F_3 using the majority function:

$$
\begin{aligned}
W_{maj} = maj(W_9, W_{30}, W_{52}) &= (W_9 \wedge W_{30}) \vee (W_9 \wedge W_{52}) \vee (W_{30} \wedge W_{52}), \\
F_1 &= W_9 \oplus \neg W_{maj}, \\
F_2 &= W_{30} \oplus \neg W_{maj}, \\
F_3 &= W_{52} \oplus \neg W_{maj}.
\end{aligned}
$$

Here all operations are bitwise operations over vectors of the length D.

To implement the conditional shifting of an LFSR one can use the bitwise counterpart of the *bitselect* function of arity 3:

$$
a, b, c \in \{0,1\};
$$

$$
BS(a,b,c) = \begin{cases} b, a = 1, \\ c, a = 0. \end{cases}
$$

If the computational architecture lacks the hardware implementation of this function, it can be emulated with the usage of the standard bitwise functions corresponding to the matching functions from the basis B:

$$BS(a,b,c) = (a \wedge b) \vee (\neg a \wedge c).$$

Table 1: Performance of two implementations of Anderson's attack (search space size is 2^{53}) on a CPU and a GPU, measured in millions of key's checks per second.

Computational device	Bit-slicing	LFSR precomputation
CPU Intel Core I7 930	37	7
GPU NVIDIA GTX 1050 Ti	9180	483
GPU NVIDIA GTX 1050 Ti (LOP3.LUT)	11950	-

Shifting LFSR1 with the use of the bitwise counterpart of $BS(a,b,c)$ and the corresponding shifting flag F_1 looks the following way (example for LFSR1):

$$W_1' = BS(F_1, (W_{19} \oplus W_{18} \oplus W_{17} \oplus W_{14}), W_1);$$
$$W_n' = BS(F_1, W_{n-1}, W_n), n \in \{2, .., 19\}.$$

Some important details about bit-slicing implementation of Anderson's attack should still be covered. This attack follows 2 steps:

1. calculation of the values of 11 bits of LFSR2 lying left of the clocking bit using the information from the guessed 53 bits and the known keystream;

2. clocking the generator as normal to check if the guessed filling of the generator matches the known keystream.

The irregular clocking of the A5/1 generator makes it generally impossible to predict how many clockings of the generator (bits of keystream) would be needed to shift 11 times LFSR2 to complete Stage 1 of the attack. Therefore, we again put to use the bitselect function to implement the split of the attack into 2 stages. Each individual thread should be able to advance from Stage 1 to Stage 2 independently of other threads. To achieve this, we introduce the special Boolean vector $\phi = (\phi_1, ..., \phi_D)$, called *attack stage flag*. The thread with number $i, i \in \{1, ..., D\}$ being in Stage 1 of the attack corresponds to $\phi_i = 0$, and Stage 2 of the attack corresponds to $\phi_i = 1$. Let $y_1, ..., y_{64}$ be the bits of the keystream analyzed. Now the shifting of LFSR2 takes into account the stage of the attack through the usage of the attack stage flag:

$$W_{41}^* = BS(\phi, W_{41}, (y \oplus W_{19} \oplus W_{64}));$$
$$W_{20}' = BS(F_2, (W_{41}^* \oplus W_{40}), W_{20});$$
$$W_n' = BS(F_2, W_{n-1}, W_n), n \in \{21, ..., 41\}.$$

Here W_{41}^* is a helper vector holding temporary data, y is the current bit of the keystream, in the form of a vector consisting of D copies of the corresponding bit of the keystream. The goal of Stage 1 is to calculate all 11 unknown bits of LFSR2 by using known keystream and guessed last bits of LFSR1 and LFSR3. At Stage 2 the whole generator's state is known, and the generator is clocked as normal. For the i-th thread the attack stage flag ϕ_i is set to 1 after LFSR2 of this thread was shifted 11 times. To count the number of LFSR2 shifts individually for each thread, the bit-slicing implementation of an incremental counter is used.

Anderson's attack described above was implemented on an NVIDIA GPU with the use of CUDA SDK 8.0[1]. The comparison of the performance of a GPU to a CPU in execution of bit-slicing and LFSR precomputation-based implementations of Anderson's attack is shown in Table 1. The last row of Table 1 corresponds to the case, in which the bitselect function is implemented using the LOP3.LUT[2] instruction.

Data provided in Table 1 tells us that even one mid-range consumer GPU is enough to make Anderson's attack runtime practical (it will take around 250 hours). A modern computational cluster outfitted with GPUs will perform the attack in mere minutes. Anderson's attack's advantage over the rainbow-tables attack [Noh10] is the former's ability to restore the secret key from 64 bits of keystream with the 100% probability. Its advantge over the attack described in [GNR08] is in the usage of a consumer-grade off-the-shelf hardware.

[1]https://developer.nvidia.com/cuda-toolkit
[2]LOP3.LUT is a special instruction that implements arbitrary bitwise functions of arity 3 in hardware.

Table 2:
Original secret keys and collisions of the A5/1 generator (in hexadecimal format).

Instance	Keystream	Secret key	
1	0x770c0410869366f1	original	0x11b8e4340276c4ee
		collision	0x42634f3266d302a3
2	0xae9590560c26e9ed	original	0x4c656fd73e59ab9b
		collision	0xcf23e4722e3cfb68
3	0xdd4b3ab7f6cf8224	original	0x09429d158555f4b3
		collision	0x09429d158553e967
		collision	0x40e5f2c8128a1781
4	0x93cd42d97eb75fd9	original	0xfa386a338355aafd
		collision	0xf9e81096bb4d0aad
		collision	0xf9e81096bb4a8556
5	0x925e423c98121152	original	0xe5cf81035ce5fbe2
6	0x3b3464bd6e377b87	original	0x9625e9d810b46248
		collision	0xf5aa1be2d6c36e18
7	0x0367d29121dd1677	original	0xd1b8b06086edf162
8	0x6b49230b7fc0249d	original	0xbe81a896968c486b
9	0xc65847556752d14c	original	0xb6f65d2855a211c0
		collision	0xb6f65d2855a508e0
10	0x07bb7f83d26072ec	original	0x122a1a2955286b9f
		collision	0xd5151aaa50490012

4 Implementation of Anderson's Attack in a volunteer computing project

In order to solve 10 cryptanalysis problems for the A5/1 keystream generator we launched the volunteer computing project AndersonAttack@home. This project is based on BOINC (Berkley Open Infrastructure for Network Compuitng [And04]). The client (computing) application of this project is based on the CUDA implementation, which was described in the previous section.

In the first stage of our experiment, a family of workunits was generated on the project server. In each workunit values of 12 out of 53 guessing bits (see Figure 2) were fixed. Thus, 40960 workunits were generated for 10 cryptanalysis problems in total. In the next stage, all generated workunits were processed in a desktop grid formed by the project's hosts. This took about 7 days. As a result, solutions for all considered problems were successfully found (see Table 2). It should be noted, that for 7 out of 10 problems the collisions were found.

Usually, the value of deadline for workunits in BOINC projects is 10-14 days. In our case, we used a deadline of 1 day, because the experiment was quite small. That is why all workunits were processed quickly. The same effect could be achieved by task scheduling (see, e.g., [MNI15]). In the considered experiment the project's performance was comparable to that of a computational cluster equipped with 30 modern GPUs. According to the BOINC statistics, 143 active hosts belonging to 90 volunteers participated in the experiment. Here by *active host* we mean a host which correctly processed at least one task.

5 Related work

It seems that the first practical attack on A5/1 was presented in [GNR08]. Its authors implemented the optimized variant of Anderson's attack on a specialized computational device of their own design, assembled from 120 "Xilinx Spartan 3" FPGAs. They state that the attack took about 6 hours. [GKN$^+$08].

In [SZBP11] using SAT-based cryptanalysis 3 instances of the A5/1 cryptanalysis were solved in a service grid. In SAT@home [ZMK$^+$16] volunteer computing project several dozens cryptanalysis problems for the A5/1 generator were solved (only one burst — 114 bits of keystream — was used each time)[SZ16].

The A5/1 Cracking Project put rainbow-tables (2 Tb in total) for A5/1 into public domain at the end of 2009 [Noh10]. By analyzing 2 frames (912 bits) of known keystream with the help of these tables one can restore the secret key with probability of success over 85%.

6 Conclusion

In this paper, we presented a GPU-based implementation of Anderson's attack on the A5/1 keystream generator. The meticulous adaptation of this attack to the SIMD architecture made it possible to solve several cryptanalysis problems in a BOINC-based desktop grid.

Acknowledgment

The authors would like to thank Stepan Kochemazov for helpful discussions and valuable comments. We also thank all volunteers, whose computers took part in the experiment.

References

[And] Ross Anderson. A5 (was: Hacking digital phones). Newsgroup Communication, 1994.

[And04] David P. Anderson. BOINC: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID'04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[Bar09] Gregory V. Bard. *Algebraic Cryptanalysis*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[BS16] Vadim Bulavintsev and Alexander Semenov. Inverting A5/1 cryptographic function on a GPU with alternative A5/1 algorithm software implementations. In *10th Annual International Scientific Conference on Parallel Computing Technologies Arkhangelsk, Russia, March 29-31, 2016*, volume 1576 of *CEUR-WS*, pages 472–481, 2016.

[BSW00] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.

[GKN+08] Tim Güneysu, Timo Kasper, Martin Novotný, Christof Paar, and Andy Rupp. Cryptanalysis with COPACOBANA. *IEEE Trans. Comput.*, 57(11):1498–1513, November 2008.

[GNR08] Timo Gendrullis, Martin Novotný, and Andy Rupp. A real-world attack breaking A5/1 within hours. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 266–282. Springer, 2008.

[MNI15] Vladimir V. Mazalov, Natalia N. Nikitina, and Evgeny E. Ivashko. Task scheduling in a desktop grid to minimize the server load. In *Proceedings of the 13th International Conference on Parallel Computing Technologies - Volume 9251*, pages 273–278, New York, NY, USA, 2015. Springer-Verlag New York, Inc.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[Noh10] Karsten Nohl. Attacking phone privacy. In *BlackHat 2010 Lecture Notes, Las-Vegas, USA, July 28-29, 2010*, pages 1–6, 2010.

[SZ16] Alexander Semenov and Oleg Zaikin. Algorithm for finding partitionings of hard variants of Boolean satisfiability problem with application to inversion of some cryptographic functions. *SpringerPlus*, 5(1):1–16, 2016.

[SZBP11] Alexander Semenov, Oleg Zaikin, Dmitry Bespalov, and Mikhail Posypkin. Parallel logical cryptanalysis of the generator A5/1 in bnb-grid system. In Victor Malyshkin, editor, *Parallel Computing Technologies - 11th International Conference, PaCT 2011, Kazan, Russia, September 19-23, 2011. Proceedings*, volume 6873 of *Lecture Notes in Computer Science*, pages 473–483. Springer, 2011.

[ZMK+16] Oleg Zaikin, Maxim Manzyuk, Stepan Kochemazov, Igor Bychkov, and Alexander Semenov. A volunteer-computing-based grid architecture incorporating idle resources of computational clusters. In Ivan Dimov, István Faragó, and Lubin G. Vulkov, editors, *Numerical Analysis and Its Applications - 6th International Conference, NAA 2016, Lozenetz, Bulgaria, June 15-22, 2016, Revised Selected Papers*, volume 10187 of *Lecture Notes in Computer Science*, pages 769–776, 2016.