

27 April 2000. Thanks to Adi Shamir.

This paper was presented at the [Fast Software Encryption Workshop 2000](#), April 10-12, 2000, New York City. It supercedes an earlier version, "Real Time Cryptanalysis of the Alleged A5/1 on a PC (preliminary draft)," by Alex Biryukov and Adi Shamir, dated December 9, 1999.

Original 18-page paper: <http://cryptome.org/a5.ps> (Postscript, 297K)

Zipped Postscript: <http://cryptome.or/a5.zip> (104K)

---

## Real Time Cryptanalysis of A5/1 on a PC

Alex Biryukov \* Adi Shamir \*\* David Wagner \*\*\*

**Abstract.** A5/1 is the strong version of the encryption algorithm used by about 130 million GSM customers in Europe to protect the over-the-air privacy of their cellular voice and data communication. The best published attacks against it require between  $2^{40}$  and  $2^{45}$  steps. This level of security makes it vulnerable to hardware-based attacks by large organizations, but not to software-based attacks on multiple targets by hackers.

In this paper we describe new attacks on A5/1, which are based on subtle flaws in the tap structure of the registers, their noninvertible clocking mechanism, and their frequent resets. After a  $2^{48}$  parallelizable data preparation stage (which has to be carried out only once), the actual attacks can be carried out in real time on a single PC.

The first attack requires the output of the A5/1 algorithm during the first two minutes of the conversation, and computes the key in about one second. The second attack requires the output of the A5/1 algorithm during about two seconds of the conversation, and computes the key in several minutes. The two attacks are related, but use different types of time-memory tradeoff. The attacks were verified with actual implementations, except for the preprocessing stage which was extensively sampled rather than completely executed.

REMARK: We based our attack on the version of the algorithm which was derived by reverse engineering an actual GSM telephone and published at <http://www.scard.org>. We would like to thank the GSM organization for graciously confirming to us the correctness of this unofficial description. In addition, we would like to stress that this paper considers the narrow issue of the cryptographic strength of A5/1, and not the broader issue of the practical security of fielded GSM systems, about which we make no claims.

\* Computer Science department, The Weizmann Institute, Rehovot 76100, Israel.

\*\* Computer Science department, The Weizmann Institute, Rehovot 76100, Israel.

\*\*\* Computer Science department, University of California, Berkeley CA 94720, USA.

---

### 1 Introduction

The over-the-air privacy of GSM telephone conversations is protected by the A5 stream cipher. This algorithm has two main variants: The stronger A5/1 version is used by about 130 million customers in Europe, while the weaker A5/2 version is used by another 100 million customers in other markets. The approximate design of A5/1 was leaked in 1994, and the exact design of both A5/1 and A5/2 was reverse engineered by Briceno from an actual GSM telephone in 1999 (see [3]).

In this paper we develop two new cryptanalytic attacks on A5/1, in which a single PC can extract the conversation key in real time from a small amount of generated output. The attacks are related, but each one of them optimizes a different parameter: The first attack (called **the biased birthday attack**) requires two minutes of data and one second of processing time, whereas the second attack (called the **the random subgraph attack**) requires two seconds of data and several minutes of processing time. There are many possible choices of tradeo parameters in these attacks, and three of them are summarized in Table 1.

Attack Type	Preprocessing steps	Available data	Number of 73GB disks	Attack time
Biased Birthday attack (1)	$2^{42}$	2 minutes	4	1 second
Biased Birthday attack (2)	$2^{48}$	2 minutes	2	1 second
Random Subgraph attack	$2^{48}$	2 seconds	4	minutes

**Table 1.** Three possible tradeoff points in the attacks on A5/1.

Many of the ideas in these two new attacks are applicable to other stream ciphers as well, and define new quantifiable measures of security.

The paper is organized in the following way: Section 2 contains a full description of the A5/1 algorithm. Previous attacks on A5/1 are surveyed in Section 3, and an informal description of the new attacks is contained in Section 4. Finally, Section 5 contains various implementation details and an analysis of the expected success rate of the attacks, based on large scale sampling with actual implementations.

## 2 Description of the A5/1 stream cipher

A GSM conversation is sent as a sequence of frames every 4.6 millisecond. Each frame contains 114 bits representing the digitized A to B communication, and 114 bits representing the digitized B to A communication. Each conversation can be encrypted by a new session key  $K$ . For each frame,  $K$  is mixed with a publicly known frame counter  $F_n$ , and the result serves as the initial state of a generator which produces 228 pseudo random bits. These bits are XOR'ed by the two parties with the 114+114 bits of the plaintext to produce the 114+114 bits of the ciphertext.

A5/1 is built from three short linear feedback shift registers (LFSR) of lengths 19, 22, and 23 bits, which are denoted by  $R_1$ ;  $R_2$  and  $R_3$  respectively. The rightmost bit in each register is labelled as bit zero. The taps of  $R_1$  are at bit positions 13,16,17,18; the taps of  $R_2$  are at bit positions 20,21; and the taps of  $R_3$  are at bit positions 7, 20,21,22 (see Figure 1). When a register is clocked, its taps are XORed together, and the result is stored in the rightmost bit of the left-shifted register.

The three registers are maximal length LFSR's with periods  $2^{19} - 1$ ,  $2^{22} - 1$ , and  $2^{23} - 1$ , respectively. They are clocked in a stop/go fashion using the following majority rule: Each register

has a single "clocking" tap (bit 8 for  $R1$ , bit 10 for  $R2$ , and bit 10 for for  $R3$ ); each clock cycle, the majority function of the clocking taps is calculated and only those registers whose clocking taps agree with the majority bit are actually clocked. Note that at each step either two or three registers are clocked, and that each register moves with probability  $3/4$  and stops with probability  $1/4$ .

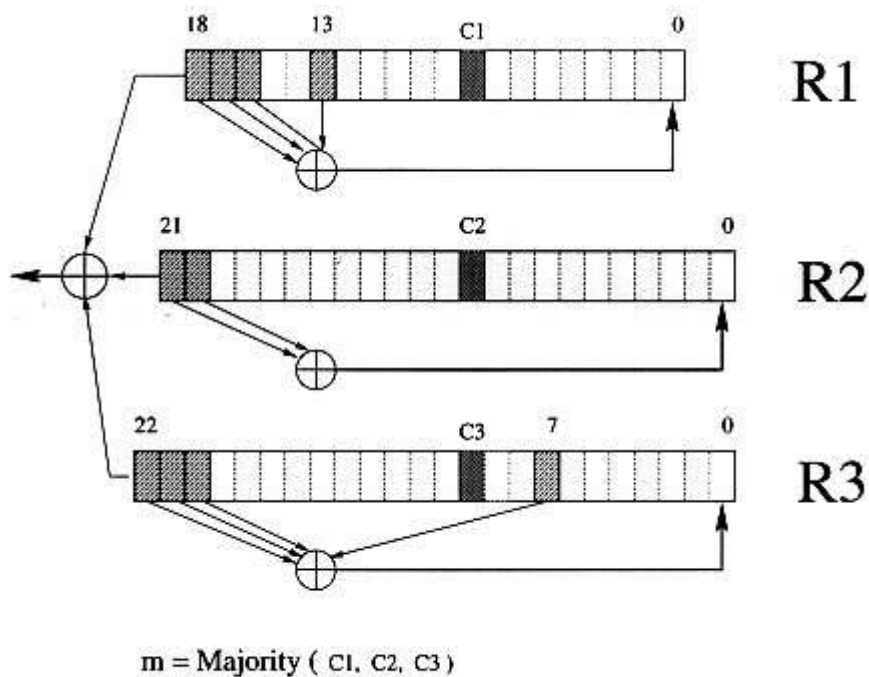


Figure 1: The A5/1 stream cipher.

The process of generating pseudo random bits from the session key  $K$  and the frame counter  $F_n$  is carried out in four steps:

- The three registers are zeroed, and then clocked for 64 cycles (ignoring the stop/go clock control). During this period each bit of  $K$  (from lsb to msb) is XOR'ed in parallel into the lsb's of the three registers.
- The three registers are clocked for 22 additional cycles (ignoring the stop/go clock control). During this period the successive bits of  $F_n$  (from lsb to msb) are again XOR'ed in parallel into the lsb's of the three registers. The contents of the three registers at the end of this step is called the initial state of the frame.
- The three registers are clocked for 100 additional clock cycles with the stop/go clock control but without producing any outputs.
- The three registers are clocked for 228 additional clock cycles with the stop/go clock control in order to produce the 228 output bits. At each clock cycle, one output bit is produced as the XOR of the msb's of the three registers.

### 3 Previous attacks

The attacker is assumed to know some pseudo random bits generated by A5/1 in some of the frames. This is the standard assumption in the cryptanalysis of stream ciphers, and we do not consider in this paper the crucial issue of how one can obtain these bits in fielded GSM systems. For the sake of simplicity, we assume that the attacker has complete knowledge of the outputs of

the A5/1 algorithm during some initial period of the conversation, and his goal is to find the key in order to decrypt the remaining part of the conversation. Since GSM telephones send a new frame every 4.6 milliseconds, each second of the conversation contains about  $2^8$  frames.

At the rump session of Crypto 99, Ian Goldberg and David Wagner announced an attack on A5/2 which requires very few pseudo random bits and just  $O(2^{16})$  steps. This demonstrated that the "export version" A5/2 is totally insecure.

The security of the A5/1 encryption algorithm was analyzed in several papers. Some of them are based on the early imprecise description of this algorithm, and thus their details have to be slightly modified. The known attacks can be summarized in the following way:

-- Briceno[3] found out that in all the deployed versions of the A5/1 algorithm, the 10 least significant of the 64 key bits were always set to zero. The complexity of exhaustive search is thus reduced to  $O(2^{54})$ .<sup>4</sup>

-- Anderson and Roe[1] proposed an attack based on guessing the 41 bits in the shorter  $R_1$  and  $R_2$  registers, and deriving the 23 bits of the longer  $R_3$  register from the output. However, they occasionally have to guess additional bits to determine the majority-based clocking sequence, and thus the total complexity of the attack is about  $O(2^{45})$ . Assuming that a standard PC can test ten million guesses per second, this attack needs more than a month to find one key.

-- Golic[4] described an improved attack which requires  $O(2^{40})$  steps. However, each operation in this attack is much more complicated, since it is based on the solution of a system of linear equations. In practice, this algorithm is not likely to be faster than the previous attack on a PC.

-- Golic[4] describes a general time-memory tradeoff attack on stream ciphers (which was independently discovered by Babbage [2] two years earlier), and concludes that it is possible to find the A5/1 key in 222 probes into random locations in a precomputed table with  $2^{42}$  128 bit entries. Since such a table requires a 64 terabyte hard disk, the space requirement is unrealistic. Alternatively, it is possible to reduce the space requirement to 862 gigabytes, but then the number of probes increases to  $O(2^{28})$ . Since random access to the fastest commercially available PC disks requires about 6 milliseconds, the total probing time is almost three weeks. In addition, this tradeoff point can only be used to attack GSM phone conversations which last more than 3 hours, which again makes it unrealistic.

---

<sup>4</sup> Our new attack is not based on this assumption, and is thus applicable to A5/1 implementations with full 64 bit keys. It is an interesting open problem whether we can speed it up by assuming that 10 key bits are zero.

## 4 Informal Description of the New Attacks

We start with an executive summary of the key ideas of the two attacks. More technical descriptions of the various steps will be provided in the next section.

**Key idea 1: Use the Golic time-memory tradeoff.** The starting point for the new attacks is the

time-memory tradeoff described in Golic[3], which is applicable to any cryptosystem with a relatively small number of internal states. A5/1 has this weakness, since it has  $n = 2^{64}$  states defined by the  $19+22+23 = 64$  bits in its three shift registers. The basic idea of the Golic time-memory tradeoff is to keep a large set  $A$  of precomputed states on a hard disk, and to consider the large set  $B$  of states through which the algorithm progresses during the actual generation of output bits. Any intersection between  $A$  and  $B$  will enable us to identify an actual state of the algorithm from stored information.

**Key idea 2: Identify states by prefixes of their output sequences.** Each state defines an infinite sequence of output bits produced when we start clocking the algorithm from that state. In the other direction, states are usually uniquely defined by the first  $\log(n)$  bits in their output sequences, and thus we can look for equality between unknown states by comparing such prefixes of their output sequences. During precomputation, pick a subset  $A$  of states, compute their output prefixes, and store the (prefix, state) pairs sorted into increasing prefix values. Given actual outputs of the A5/1 algorithm, extract all their (partially overlapping) prefixes, and define  $B$  as the set of their corresponding (unknown) states. Searching for common states in  $A$  and  $B$  can be efficiently done by probing the sorted data  $A$  on the hard disk with prefix queries from  $B$ .

**Key idea 3: A5/1 can be efficiently inverted.** As observed by Golic, the state transition function of A5/1 is not uniquely invertible: The majority clock control rule implies that up to 4 states can converge to a common state in one clock cycle, and some states have no predecessors. We can run A5/1 backwards by exploring the tree of possible predecessor states, and backtracking from dead ends. The average number of predecessors of each node is 1, and thus the expected number of vertices in the first  $k$  levels of each tree grows only linearly in  $k$  (see[3]). As a result, if we find a common state in the disk and data, we can obtain a small number of candidates for the initial state of the frame. The weakness we exploit here is that due to the frequent reinitializations there is a very short distance from intermediate states to initial states.

**Key idea 4: The key can be extracted from the initial state of any frame.** Here we exploit the weakness of the A5/1 key setup routine. Assume that we know the state of A5/1 immediately after the key and frame counter were used, and before the 100 mixing steps. By running backwards, we can eliminate the effect of the known frame counter in a unique way, and obtain 64 linear combinations of the 64 key bits. Since the tree exploration may suggest several keys, we can choose the correct one by mixing it with the next frame counter, running A5/1 forward for more than 100 steps, and comparing the results with the actual data in the next frame.

**Key idea 5: The Golic attack on A5/1 is marginally impractical.** By the well known birthday paradox,  $A$  and  $B$  are likely to have a common state when their sizes  $a$  and  $b$  satisfy  $a * b \approx n$ . We would like  $a$  to be bounded by the size of commercially available PC hard disks, and  $b$  to be bounded by the number of overlapping prefixes in a typical GSM telephone conversation.

Reasonable bounds on these values (justified later in this paper) are  $a \approx 2^{35}$  and  $b \approx 2^{22}$ . Their product is  $2^{57}$ , which is about 100 times smaller than  $n = 2^{64}$ . To make the intersection likely, we either have to increase the storage requirement from 150 gigabytes to 15 terabytes, or to increase the length of the conversation from two minutes to three hours. Neither approach seems to be practical, but the gap is not huge and a relatively modest improvement by two orders of magnitude is all we need to make it practical.

**Key idea 6: Use special states.** An important consideration in implementing time-memory tradeoff attacks is that access to disk is about a million times slower than a computational step, and thus it is crucial to minimize the number of times we look for data on the hard disk. An old idea due to Ron Rivest is to keep on the disk only special states which are guaranteed to produce output bits starting with a particular pattern  $\alpha$  of length  $k$ , and to access the disk only when we

encounter such a prefix in the data. This reduces the number  $b$  of disk probes by a factor of about  $2k$ . The number of points  $a$  we have to memorize remains unchanged, since in the formula  $a * b \approx n$  both  $b$  and  $n$  are reduced by the same factor  $2k$ . The downside is that we have to work  $2k$  times harder during the preprocessing stage, since only  $2^{-k}$  the preprocessing time by a factor of about 64,000, which makes it impractically long.

**Key idea 7: Special states can be efficiently sampled in A5/1.** A major weakness of A5/1 which we exploit in both attacks is that it is easy to generate all the states which produce output sequences that start with a particular  $k$ -bit pattern  $alpha$  with  $k = 16$  without trying and discarding other states. This is due to a poor choice of the clocking taps, which makes the register bits that affect the clock control and the register bits that affect the output unrelated for about 16 clock cycles, so we can choose them independently. This easy access to special states does not happen in good block ciphers, but can happen in stream ciphers due to their simpler transition functions. In fact, the maximal value of  $k$  for which special states can be sampled without trial and error can serve as a new security measure for stream ciphers, which we call its sampling resistance. As demonstrated in this paper, high values of  $k$  can have a big impact on the efficiency of time-memory tradeoff attacks on such cryptosystems.

**Key idea 8: Use biased birthday attacks.** The main idea of the first attack is to consider sets  $A$  and  $B$  which are not chosen with uniform probability distribution among all the possible states. Assume that each state  $s$  is chosen for  $A$  with probability  $P_A(s)$ , and is chosen for  $B$  with probability  $P_B(s)$ . If the means of these probability distributions are  $a/n$  and  $b/n$ , respectively, then the expected size of  $A$  is  $a$ , and the expected size of  $B$  is  $b$ .

The birthday threshold happens when  $\sum_s P_A(s) P_B(s) \approx 1$ . For independent uniform distributions, this evaluates to the standard condition  $a * b \approx n$ . However, in the new attack we choose states for the disk and states in the data with two non-uniform probability distributions which have strong positive correlation. This makes our time memory tradeoff much more efficient than the one used by Golic. This is made possible by the fact that in A5/1, the initial state of each new frame is rerandomized very frequently with different frame counters.

**Key idea 9: Use Hellman's time-memory tradeoff on a subgraph of special states.** The main idea of the second attack (called the random subgraph attack) is to make most of the special states accessible by simple computations from the subset of special states which are actually stored in the hard disk. The first occurrence of a special state in the data is likely to happen in the first two seconds of the conversation, and this single occurrence suffices in order to locate a related special state in the disk even though we are well below the threshold of either the normal or the biased birthday attack. The attack is based on a new function  $f$  which maps one special state into another special state in an easily computable way. This  $f$  can be viewed as a random function over the subspace of  $2^{48}$  special states, and thus we can use Hellman's time-memory tradeoff [4] in order to invert it efficiently. The inverse function enables us to compute special states from output prefixes even when they are not actually stored on the hard disk, with various combinations of time  $T$  and memory  $M$  satisfying  $M$  square root  $T = 2^{48}$ . If we choose  $M = 2^{36}$ , we get  $T = 2^{24}$ , and thus we can carry out the attack in a few minutes, after a  $2^{48}$  preprocessing stage which explores the structure of this function  $f$ .

**Key idea 10: A5/1 is very efficient on a PC.** The A5/1 algorithm was designed to be efficient in hardware, and its straightforward software implementation is quite slow. To execute the preprocessing stage, we have to run it on a distributed network of PC's up to  $2^{48}$  times, and thus we need an extremely efficient way to compute the effect of one clock cycle on the three registers.

We exploit the following weakness in the design of A5/1: Each one of the three shift registers is so small that we can precompute all its possible states, and keep them in RAM as three cyclic arrays, where successive locations in each array represent successive states of the corresponding shift register. In fact, we don't have to keep the full states in the arrays, since the only information we have to know about a state is its clocking tap and its output tap. A state can thus be viewed as a triplet of indices  $(i; j; k)$  into three large single bit arrays (see Figure 2).  $A_1(i); A_2(j); A_3(k)$  are the clocking taps of the current state, and  $A_1(i - 11), A_2(j - 12), A_3(k - 13)$  are the output taps of the current state (since these are the corresponding delays in the movement of clocking taps to output taps when each one of the three registers is clocked). Since there is no mixing of the values of the three registers, their only interaction is in determining which of the three indices should be incremented by 1. This can be determined by a precomputed table with three input bits (the clocking taps) and three output bits (the increments of the three registers). When we clock A5/1 in our software implementation, we don't shift registers or compute feedbacks - we just add a 0/1 vector to the current triplet of indices. A typical two dimensional variant of such movement vectors in triplet space is described in Figure 3. Note the local tree structure determined by the deterministic forward evaluation and the nondeterministic backward exploration in this triplet representation.

Since the increment table is so small, we can expand the A tables from bits to bytes, and use a larger precomputed table with  $2^{24}$  entries, whose inputs are the three bytes to the right of the clocking taps in the three registers, and outputs are the three increments to the indices which allow us to jump directly to the state which is 8 clock cycles away. The total amount of RAM needed for the state arrays and precomputed movement tables is less than 128 MB, and the total cost of advancing the three registers for 8 clock cycles is one table lookup and three integer additions! A similar table lookup technique can be used to compute in a single step output bytes instead of output bits, and to speed up the process of running A5/1 backwards.

## 5 Detailed Description of the Attacks

In this section we fill in the missing details, and analyse the success rate of the new attacks.

### 5.1 Efficient Sampling of Special States

Let *alpha* be any 16 bit pattern of bits. To simplify the analysis, we prefer to use an *alpha* which does not coincide with shifted versions of itself (such as *alpha* = 1000...0) since this makes it very unlikely that a single 228-bit frame contains more than one occurrence of *alpha*.

The total number of states which generate an output prefix of *alpha* is about  $2^{64} * 2^{-16} = 2^{48}$ . We would like to generate all of them in a (barely doable)  $2^{48}$  preprocessing stage, without trying all the  $2^{64}$  possible states and discarding the vast majority which fail the test. The low sampling resistance of A5/1 is made possible by several flaws in its design, which are exploited in the following algorithm:

-- Pick an arbitrary 19-bit value for the shortest register *R1*. Pick arbitrary values for the rightmost 11 bits in *R2* and *R3* which will enter the clock control taps in the next few cycles. We can thus define  $2^{19+11+11} = 2^{41}$  partial states.

-- For each partial state we can uniquely determine the clock control of the three registers for the next few cycles, and thus determine the identity of the bits that enter their msb's and affect the output.

-- Due to the majority clock control, at least one of  $R2$  and  $R3$  shifts a new (still unspecified) bit into its msb at each clock cycle, and thus we can make sure that the computed output bit has the desired value. Note that about half the time only one new bit is shifted (and then its choice is forced), and about half the time two new bits are shifted (and then we can choose them in two possible ways). We can keep this process alive without time consuming trial and error as long as the clock control taps contain only known bits whereas the output taps contain at least one unknown bit. A5/1 makes this very easy, by using a single clocking tap and placing it in the middle of tap of each register: We can place in  $R2$  and  $R3$  11 specified bits to the right of the clock control tap, and 11-12 unspecified bits to the right of the output tap. Since each register moves only  $3/4$  of the time, we can keep this process alive for about 16 clock cycles, as desired.

-- This process generates only special states, and cannot miss any special state (if we start the process with its partial specification, we cannot get into an early contradiction). We can similarly generate any number  $c < 2^{48}$  of randomly chosen special states in time proportional to  $c$ . As explained later in the paper, this can make the preprocessing faster, at the expense of other parameters in our attack.

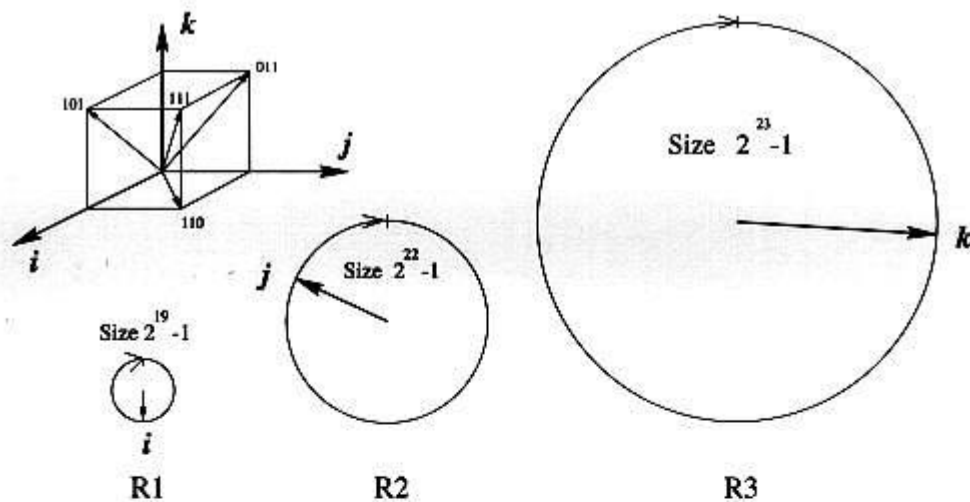


Figure 2: Triplet representation of a state.

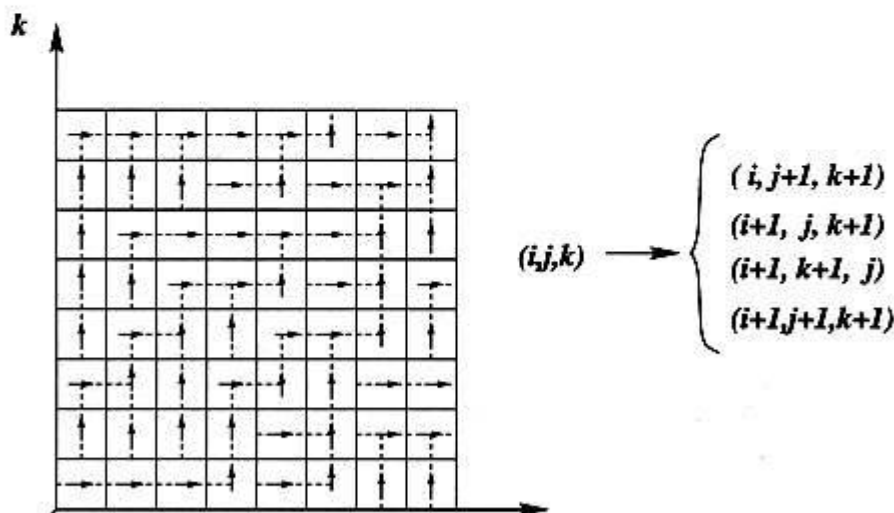






Figure 3: The state-transition graph in the triplet representation of A5/1.

## 5.2 Efficient Disk Probing

To leave room for a sufficiently long identifying prefix of 35 bits after the 16-bit *alpha*, we allow it to start only at bit positions 1 to 177 in each one of the given frames (i.e., at a distance of 101 to 277 from the initial state). The expected number of occurrences of *alpha* in the data produced by A5/1 during a two minute conversation is thus  $2^{-16} * 177 * 120 * 1000/4.6 \approx 71$ . This is the expected number of times *b* we access the hard disk. Since each random access takes about 6 milliseconds, the total disk access time becomes negligible (about 0.4 seconds).

## 5.3 Efficient Disk Storage

The data items we store on the disk are (prefix, state) pairs. The state of A5/1 contains 64 bits, but we keep only special states and thus we can encode them efficiently with shorter 48 bit names, by specifying the 41 bits of the partial state and the  $\approx 7$  choice bits in the sampling procedure. We can further reduce the state to less than 40 bits (5 bytes) by leaving some of the 48 bits unspecified. This saves a considerable fraction of the disk space prepared during preprocessing, and the only penalty is that we have to try a small number of candidate states instead of one candidate state for each one of the 71 relevant frames. Since this part is so fast, even in its slowed down version it takes less than a second.

The output prefix produced from each special state is nominally of length  $16+35=51$  bits. However, the first 16 bits are always the constant *alpha*, and the next 35 bits are stored in sorted order on the disk. We can thus store the full value of these 35 bits only once per sector, and encode on the disk only their small increments (with a default value of 1). Other possible implementations are to use the top parts of the prefixes as direct sector addresses or as file names. With these optimizations, we can store each one of the sorted (prefix, state) pairs in just 5 bytes. The largest commercially available PC hard disks (such as IBM Ultrastar 72 ZX or Seagate Cheetah 73) have 73 gigabytes. By using two such disks, we can store  $146 * 2^{30} \approx 2^{35}$  pairs during the preprocessing stage, and characterize each one of them by the (usually unique) 35-bit output prefix which follows *alpha*.

## 5.4 Efficient Tree Exploration

The forward state-transition function of A5/1 is deterministic, but in the reverse direction we have to consider four possible predecessors. About  $3/8$  of the states have no predecessors,  $13/32$  of the states have one predecessor,  $3/32$  of the states have two predecessors,  $3/32$  of the states have three predecessors, and  $1/32$  of the states have four predecessors.

Since the average number of predecessors is 1, Golic assumed that a good statistical model for the generated trees of predecessors is the critical branching process (see [3]). We were surprised to discover that in the case of A5/1, there was a very significant difference between the predictions of this model and our experimental data. For example, the theory predicted that only 2% of the states would have some predecessor at depth 100, whereas in a large sample of 100,000,000 trees we generated from random A5/1 states the percentage was close to 15%. Another major difference was found in the tail distributions of the number of sons at depth 100: Theory predicted that in our