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 tradeoffs. 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.

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]).

* 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.

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 tradeoff 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 $R1, R2$ and $R3$ respectively. The rightmost bit in each register is labelled as bit zero. The taps of $R1$ are at bit positions 13,16,17,18; the taps of $R2$ are at bit positions 20,21; and the taps of $R3$ 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.

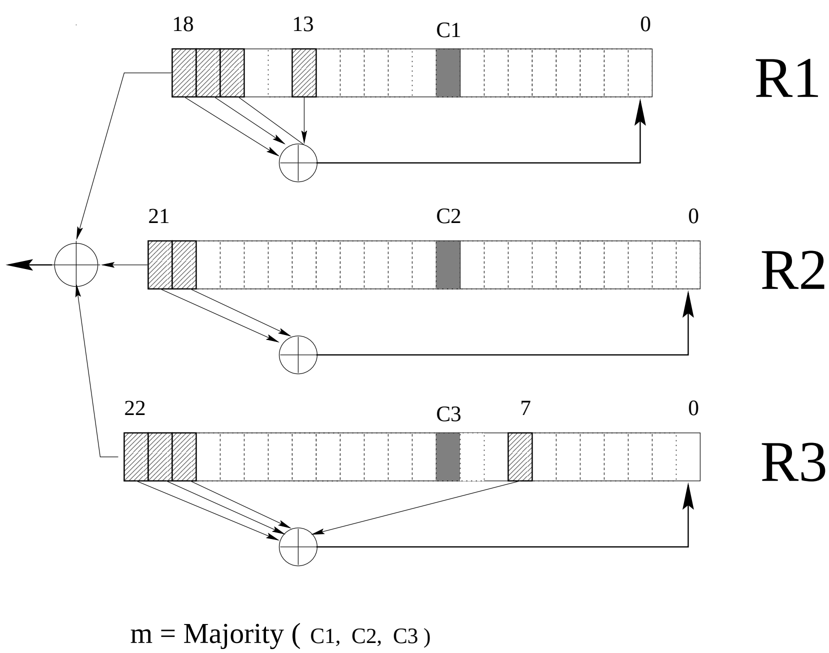

Fig. 1. The A5/1 stream cipher.

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 $O(2^{54})$. [4]
- 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.

[4] 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.

– 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 2^{22} 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.

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 α 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 2^k. 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 2^k. The downside is that we have to work 2^k times harder during the preprocessing stage, since only 2^{-k} of the random states we try produce outputs with such a k bit prefix. If we try to reduce the number of disk access steps in the time memory attack on A5/1 from 2^{22} to 2^6, we have to increase 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 α 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 $M\sqrt{T} = 2^{48}$. 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 α be any 16 bit pattern of bits. To simplify the analysis, we prefer to use an α 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 α.

The total number of states which generate an output prefix of α 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:

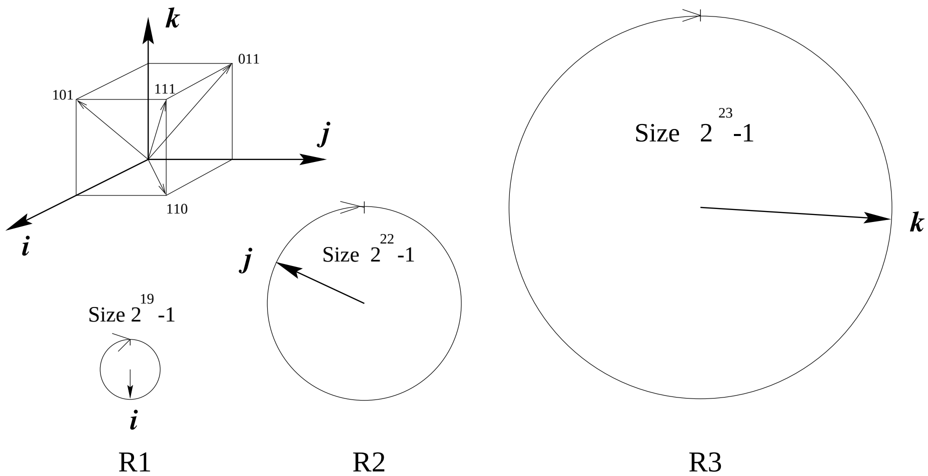

Fig. 2. Triplet representation of a state.

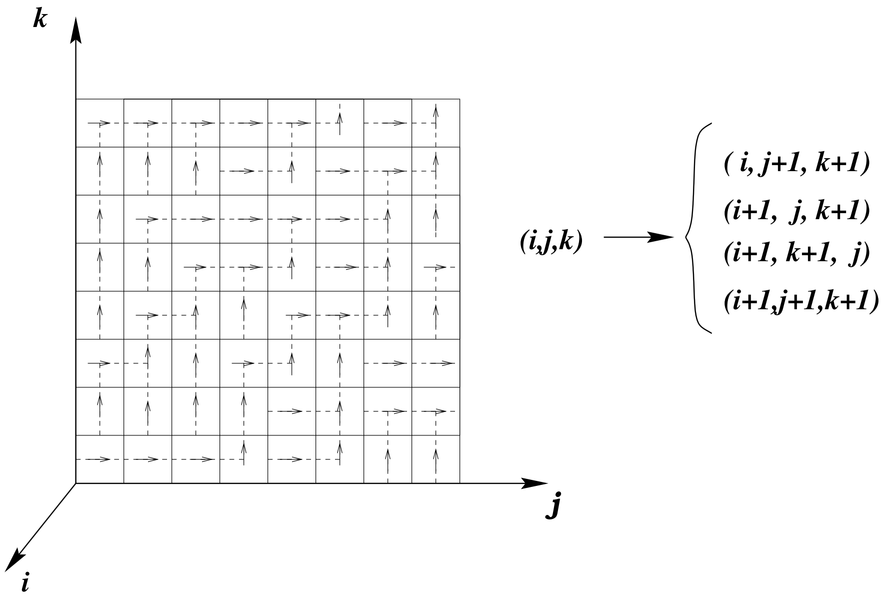

Fig. 3. The state-transition graph in the triplet representation of A5/1.

- 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 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.

5.2 Efficient Disk Probing

To leave room for a sufficiently long identifying prefix of 35 bits after the 16-bit α, 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 α 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 ≈ 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 α, 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 $146 * 2^{30}/5 \approx 2^{35}$ pairs during the preprocessing stage, and characterize each one of them by the (usually unique) 35-bit output prefix which follows α.

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 sample we should see some cases with close to 1000 sons, whereas in our sample we never saw trees with more than 120 sons at depth 100.

5.5 The Biased Birthday Attack.

To analyse the performance of our biased birthday attack, we introduce the following notation:

Definition 1 *A state s is coloured* **red**, *if the sequence of output bits produced from state s starts with α (i.e., it is a special state). The subspace of all the red states is denoted by R.*

Definition 2 *A state is coloured* **green**, *if the sequence of output bits produced from state s contains an occurrence of α which starts somewhere between bit positions 101 and 277. The subspace of all the green states is denoted by G.*

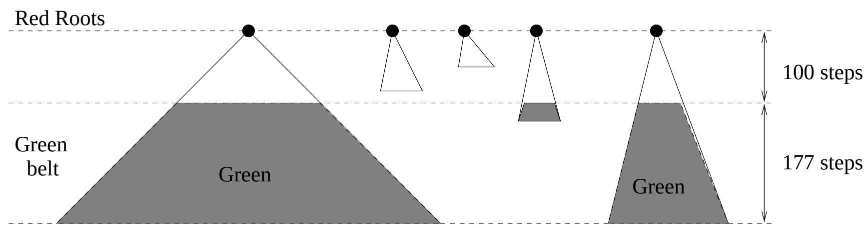

Fig. 4. Trees of different sizes.

The red states are the states that we keep in the disk, look for in the data, and try to collide by comparing their prefixes. The green states are all the states that could serve as initial states in frames that contain α. Non-green initial states are of no interest to us, since we discard the frames they generate from the actual data.

The size of R is approximately 2^{48}, since there are 2^{64} possible states, and the probability that α occurs right at the beginning of the output sequence is 2^{-16}. Since the redness of a state is not directly related to its separate coordinates i, j, k in the triplet space, the red states can be viewed as randomly and sparsely located in this representation. The size of G is approximately $177 * 2^{48}$ (which is still a small fraction of the state space) since α has 177 opportunities to occur along the output sequence.

Since a short path of length 277 in the output sequence is very unlikely to contain two occurrences of α, the relationship between green and red states is essentially many to one: The set of all the relevant states we consider can be viewed as a collection of disjoint trees of various sizes, where each tree has a red state as its root and a "belt" of green states at levels 101 to 277 below it (see Figure 4). The weight $W(s)$ of a tree whose root is the red state s is defined as the number of green states in its belt, and s is called k-heavy if $W(s) \geq k$.

The crucial observation which makes our biased birthday attack efficient is that in A5/1 there is a huge variance in the weights of the various red states. We ran the tree exploration algorithm on 100,000,000 random states and computed their weights. We found out that the weight of about 85% of the states was zero, because their trees died out before reaching depth 100. Other weights ranged all the way from 1 to more than 26,000.

The leftmost graph of Figure 5 describes for each x which is a multiple of 100 the value y which is the total weight of all the trees whose weights were between x and $x + 100$. The total area under the graph to the right of $x = k$ represents the total number of green states in all the k-heavy trees in our sample.

The initial mixing of the key and frame number, which ignores the usual clock control and flips the least significant bits of the registers about half the time before shifting them, can be viewed as random jumps with uniform probability distribution into new initial states: even a pair of frame counters with Hamming distance 1 can lead to far away initial states in the triplet space. When we restrict our attention to the frames that contain α, we get a uniform probability distribution over the green states, since only green states can serve as initial states in such frames.

The red states, on the other hand, are not encountered with uniform probability distribution in the actual data. For example, a red state whose tree has no green belt will never be seen in the data. On the other hand, a red state with a huge green belt has a huge number of chances to be reached when the green initial state is chosen with uniform probability distribution. In fact the probability of encountering a particular red state s in a particular frame which is known to contain α is the ratio of its weight $W(s)$ and the total number of green states $177 * 2^{48}$, and the probability of encountering it in one of the 71 relevant frames is $P_B(s) = 71 * W(s)/(177 * 2^{48})$.

Since $P_B(s)$ has a huge variance, we can maximize the expected number of collisions $\sum_s P_A(s) * P_B(s)$ by choosing red points for the hard disk not with uniform probability distribution, but with a biased probability $P_A(s)$ which maximizes the correlation between these distributions, while minimizing the expected size of A. The best way to do this is to keep on the disk only the heaviest trees. In other words, we choose a threshold number k, and define $P_A(s) = 0$ if $W(s) < k$, and $P_A(s) = 1$ if $W(s) \geq k$. We can now easily compute the expected number of collisions by the formula:

$$\sum_s P_A(s) * P_B(s) = \sum_{s|W(s)\geq k} 71 * W(s)/(177 * 2^{48})$$

which is just the number of red states we keep on the disk, times the average weight of their trees, times $71/(177 * 2^{48})$.

In our actual attack, we keep 2^{35} red states on the disk. This is a 2^{-13} fraction of the 2^{48} red states. With such a tiny fraction, we can choose particularly heavy trees with an average weight of 12,500. The expected number of colliding red states in the disk and the actual data is $2^{35} * 12,500 * 71/(177 * 2^{48}) \approx 0.61$. This expected value makes it quite likely that a collision will actually exist. [5]

The intuition behind the biased time memory tradeoff attack is very simple. We store red states, but what we really want to collide are the green states in their belts (which are accessible from the red roots by an easy computation). The 71 green states in the actual data are uniformly distributed, and thus we want to cover about 1% of the green area under the curve in the right side of Figure 5. Standard time memory tradeoff attacks store random red states, but each stored state increases the coverage by just 177 green states on average. With our optimized choice in the preprocessing stage, each stored state increases the coverage by 12,500 green states on average, which improves the efficiency of the attack by almost two orders of magnitude.

[5] Note that in time memory tradeoff attacks, it becomes increasingly expensive to push this probability towards 1, since the only way to guarantee success is to memorize the whole state space.

5.6 Efficient Determination of Initial States

One possible disadvantage of storing heavy trees is that once we find a collision, we have to try a large number of candidate states in the green belt of the colliding red state. Since each green state is only partially specified in our compact 5-byte representation, the total number of candidate green states can be hundreds of thousands, and the real time part of the attack can be relatively slow.

However, this simple estimate is misleading. The parasitic red states obtained from the partial specification can be quickly discarded by evaluating their outputs beyond the guaranteed occurrence of α and comparing it to the bits in the given frame. In addition, we know the exact location of α in this frame, and thus we know the exact depth of the initial state we are interested in within the green belt. As a result, we have to try only about 70 states in a cut through the green belt, and not the 12,500 states in the full belt.

5.7 Reducing the Preprocessing Time of the Biased Birthday Attack

The 2^{48} complexity of the preprocessing stage of this attack can make it too time consuming for a small network of PC's. In this section we show how to reduce this complexity by any factor of up to 1000, by slightly increasing either the space complexity or the length of the attacked conversation.

The efficient sampling procedure makes it possible to generate any number $c < 2^{48}$ of random red states in time proportional to c. To store the same number of states in the disk, we have to choose a larger fraction of the tested trees, which have a lower average weight, and thus a less efficient coverage of the green states. Table 2 describes the average weight of the heaviest trees for various fractions of the red states, which was experimentally derived from our sample of 100,000,000 A5/1 trees. This table can be used to choose the appropriate value of k in the

Average Weights							
2^{-4}	2432	2^{-5}	3624	2^{-6}	4719	2^{-7}	5813
2^{-8}	6910	2^{-9}	7991	2^{-10}	9181	2^{-11}	10277
2^{-12}	11369	2^{-13}	12456	2^{-14}	13471	2^{-15}	14581
2^{-16}	15686	2^{-17}	16839	2^{-18}	17925	2^{-19}	19012
2^{-20}	20152	2^{-21}	21227	2^{-22}	22209	2^{-23}	23515
2^{-24}	24597	2^{-25}	25690	2^{-26}	26234		

Table 2. The average weight of the heaviest trees for various fractions of R.

definition the k-heavy trees for various choices of c. The implied tradeoff is very favorable: If we increase the fraction from 2^{-13} to 2^{-7}, we can reduce the preprocessing time by a factor of 64 (from 2^{48} to 2^{42}), and compensate by either doubling the length of the attacked conversation from 2 minutes to 4 minutes, or doubling the number of hard disks from 2 to 4. The extreme point in this tradeoff is to store in the disk all the sampled red states with nonzero weights (the other sampled red states are just a waste of space, since they will never be seen in the actual data). In A5/1 about 15% of the red states have nonzero weights, and thus we have to sample about 2^{38} red states in the preprocessing stage in order to find the 15% among them (about 2^{35} states) which we want to store, with an average tree weight of 1180. To keep the same probability of success, we have to attack conversations which last about half an hour.

A further reduction in the complexity of the preprocessing stage can be obtained by the early abort strategy: Explore each red state to a shallow depth, and continue to explore only the most promising candidates which have a large number of sons at that depth. This heuristic does not guarantee the existence of a large belt, but there is a clear correlation between these events.

To check whether the efficiency of our biased birthday attack depends on the details of the stream cipher, we ran several experiments with modified variants of A5/1. In particular, we concentrated on the effect of the clock control rule, which determines the noninvertibility of the model. For example, we hashed the full state of the three registers and used the result to choose among the four possible majority-like movements (+1,+1,+1), (+1,+1,0), (+1,0,+1), (0,+1,+1) in the triplet space. The results were very different from the real majority rule. We then replaced the majority rule by a minority rule (if all the clocking taps agree, all the registers move, otherwise only the minority register moves). The results of this minority rule were very similar to the majority-like hashing case, and very different from the real majority case (see Figure 5). It turns out that in this sense A5/1 is actually stronger than its modified versions, but we do not currently understand the reason for this strikingly different behavior. We believe that the type of data in Table 2, which we call the **tail coverage** of the cryptosystem, can serve as a new security measure for stream ciphers with noninvertible state transition functions.

5.8 Extracting the Key From a Single Red State

The biased birthday attack was based on a direct collision between a state in the disk and a state in the data, and required ≈ 71 red states from a relatively long (≈ 2 minute) prefix of the conversation. In the random subgraph attack we use indirect collisions, which make it possible to find the key with reasonable probability from the very first red state we encounter in the data, even though it is unlikely to be stored in the disk. This makes it possible to attack A5/1 with less than two seconds of available data. The actual attack requires several minutes instead of one second, but this is still a real time attack on normal telephone conversations.

The attack is based on Hellman's original time-memory tradeoff for block ciphers, described in [4]. Let E be an arbitrary block cipher, and let P be some fixed plaintext. Define the function f from keys K to ciphertexts C by $f(K) = E_K(P)$. Assuming that all the plaintexts, ciphertexts and keys have the same binary size, we can consider f as a random function (which is not necessarily one-to-one) over a common space U. This function is easy to evaluate and to iterate but difficult to invert, since computing the key K from the ciphertext $f(K) = E_K(P)$ is essentially the problem of chosen message cryptanalysis.

Hellman's idea was to perform a precomputation in which we choose a large number m of random start points in U, and iterate f on each one of them t times. We store the m (start point, end point) pairs on a large disk, sorted into increasing endpoint order. If we are given $f(K)$ for some unknown K which is located somewhere along one of the covered paths, we can recover K by repeatedly applying f in the easy forward direction until we hit a stored end point, jump to its corresponding start point, and continue to apply f from there. The last point before we hit $f(K)$ again is likely to be the key K which corresponds to the given ciphertext $f(K)$.

Since it is difficult to cover a random graph with random paths in an efficient way, Hellman proposed a rerandomization technique which creates multiple variants of f (e.g., by permuting the order of the output bits of f). We use t variants f_i, and iterate each one of them t times on m random start points to get m corresponding end points. If the parameters m and t satisfy $mt^2 = |U|$, then each state is likely to be covered by one of the variants of f. Since we have to handle each variant separately (both in the preprocessing and in the actual attack), the total memory becomes $M = mt$ and the total running time becomes $T = t^2$, where M and T can be anywhere along the tradeoff curve $M\sqrt{T} = |U|$. In particular, Hellman suggests using $M = T = |U|^{2/3}$.

A straightforward application of this $M\sqrt{T} = |U|$ tradeoff to the $|U| = 2^{64}$ states of A5/1 with the maximal memory $M = 2^{36}$ requires time $T = 2^{56}$, which is much worse than previously known attacks. The basic idea of the new random subgraph attack is to apply the time-memory tradeoff to the subspace R of 2^{48} red states, which is made possible by the fact that it can be efficiently sampled. Since T occurs in the tradeoff formula $M\sqrt{T} = |U|$ with a square root, reducing the size of the graph by a modest 2^{16} (from $|U| = 2^{64}$ to $|R| = 2^{48}$) and using the same memory ($M = 2^{36}$), reduces the time by a huge factor of 2^{32} (from $T = 2^{56}$ to just $T = 2^{24}$). This number of steps can be carried out in several minutes on a fast PC.

What is left is to design a random function f over R whose output-permuted variants are easy to evaluate, and for which the inversion of any variant yields the desired key. Each state has a "full name" of 64 bits which describes the contents of its three registers. However, our efficient sampling technique enables us to give each red state a "short name" of 48 bits (which consists of the partial contents of the registers and the random choices made during the sampling process), and to quickly translate short names to full names. In addition, red states are characterized (almost uniquely) by their "output names" defined as the 48 bits which occur after α in their output sequences. We can now define the desired function f over 48-bit strings as the mapping from short names to output names of red states: Given a 48-bit short name x, we expand it to the full name of a red state, clock this state 64 times, delete the initial 16-bit α, and define $f(x)$ as the remaining 48 output bits. The computation of $f(x)$ from

x can be efficiently done by using the previously described precomputed tables, but the computation of x from $f(x)$ is exactly the problem of computing the (short) name of an unknown red state from the 48 output bits it produces after α. When we consider some output-permuted variant f_i of f, we obviously have to apply the same permutation to the given output sequence before we try to invert f_i over it.

The recommended preprocessing stage stores 2^{12} tables on the hard disk. Each table is defined by iterating one of the variants f_i 2^{12} times on 2^{24} randomly chosen 48-bit strings. Each table contains 2^{24} (start point, end point) pairs, but implicitly covers about 2^{36} intermediate states. The collection of all the 2^{12} tables requires 2^{36} disk space, but implicitly covers about 2^{48} red states.

The simplest implementation of the actual attack iterates each one of the 2^{12} variants of f separately 2^{12} times on appropriately permuted versions of the single red state we expect to find in the 2 seconds of data. After each step we have to check whether the result is recorded as an end point in the corresponding table, and thus we need $T = 2^{24}$ probes to random disk locations. At 6 ms per probe, this requires more than a day. However, we can again use Rivest's idea of special points: We say that a red state is bright if the first 28 bits of its output sequence contain the 16-bit α extended by 12 additional zero bits. During preprocessing, we pick a random red start point, and use f_i to quickly jump from one red state to another. After approximately 2^{12} jumps, we expect to encounter another bright red state, at which we stop and store the pair of (start point, end point) in the hard disk. In fact, each end point consists of a 28 bit fixed prefix followed by 36 additional bits. As explained in the previous attack, we do not have to store either the prefix (which is predictable) or the suffix (which is used as an index) on the hard disk, and thus we need only half the expected storage. We can further reduce the required storage by using the fact that the bright red states have even shorter short names than red states (36 instead of 48 bits), and thus we can save 25% of the space by using bright red instead of red start points in the table. [6] During the actual attack, we find the first red state in the data, iterate each one of the 2^{12} variants of f over it until we encounter a bright red state, and only then search this state among the pairs stored in the disk. We thus have to probe the disk only once in each one of the $t = 2^{12}$ tables, and the total probing time is reduced to 24 seconds.

There are many additional improvement ideas and implementation details which will be described in the final version of this paper.

Acknowledgements: We would like to thank Ross Anderson, Mike Roe, Jovan Golic, Marc Briceno, and Ian Goldberg for their pioneering contributions

[6] Note that we do not know how to jump in a direct way from one bright red state to another, since we do not know how to sample them in an efficient way. We have to try about 2^{12} red states in order to find one bright red start point, but the total time needed to find the 2^{36} bright red start points in all the tables is less than the 2^{48} complexity of the path evaluations during the preprocessing stage.

to the analysis of A5/1, which made this paper possible. We would like to thank Marc Briceno, Ian Goldberg, Ron Rivest and Nicko van Someren for sending very useful comments on the early version of this paper.

References

1. R. Anderson, M. Roe, *A5*, `http://jya.com/crack-a5.htm`, 1994.
2. S. Babbage, *A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers*, European Convention on Security and Detection, IEE Conference publication, No. 408, May 1995.
3. M. Briceno, I. Goldberg, D. Wagner, A pedagogical implementation of A5/1, `http://www.scard.org`, May 1999.
4. J. Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, proceedings of EUROCRYPT'97, LNCS 1233,pp.239–255, Springer-Verlag 1997.
5. M. E. Hellman, *A Cryptanalytic Time-Memory Trade-Off*, IEEE Transactions on Information Theory, Vol. IT-26, N 4, pp.401–406, July 1980.

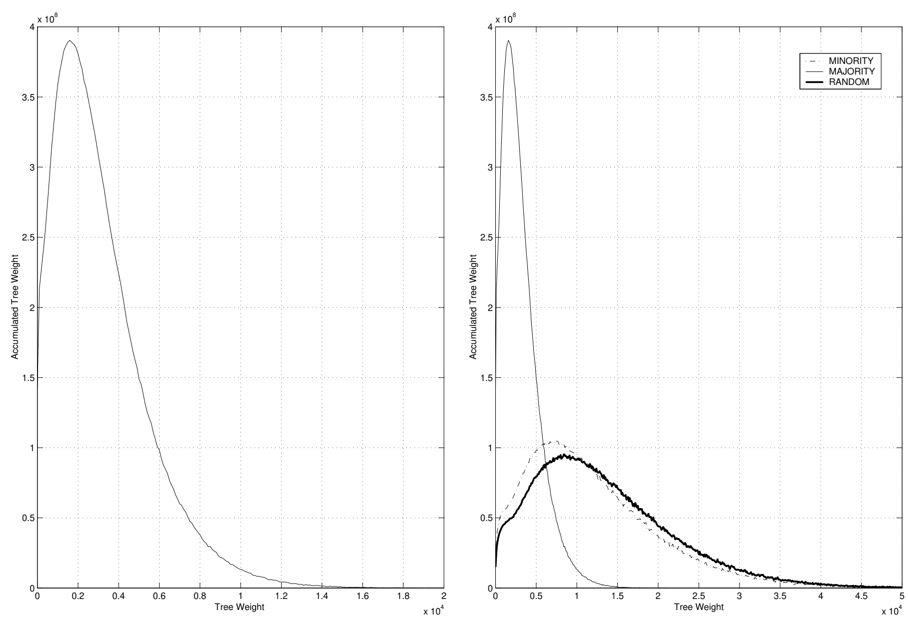

Fig. 5. Weight distributions. The graph on the left shows weight distribution for the majority function; the graph on the right compares the weight distributions of several clock-control functions.

This article was processed using the LaTeX macro package with LLNCS style