

Cryptanalysis of the GSM Algorithms

Abstract

The goal of the project is to investigate the cryptographic strength of the GSM encryption algorithms. The three core GSM algorithms are:

- A3 authentication algorithm
- A5/1 "strong" over-the-air voice-privacy algorithm
- A8 voice-privacy key generation algorithm

The Five finalists in the Advanced Encryption standard were also investigated.

Declaration

This report is presented in partial fulfilment of the requirements for the final year project for the Computer Engineering Degree.

It is my own work, and where use has been made of the work of other people it has been fully acknowledged and fully referenced.

Signature: _____

Eoin Ward

Date:

TABLE OF CONTENTS

[TABLE OF CONTENTS..... *](#)

1 INTRODUCTION *

1.1 Permutation and Substitution Boxes *

1.1.1 P-Boxes *

1.1.2 S-boxes *

1.1.3 Substitution-Permutation Network *

1.2 Algorithms *

1.2.1 Symmetric Algorithms *

1.2.2 Block Ciphers *

1.2.3 Public-Key Algorithms *

1.2.4 Stream Ciphers *

1.3 Cryptanalysis *

1.3.1 Kerckhoff's Principle. *

1.3.2 Work Factor *

1.3.3 Differential Cryptanalysis *

1.3.4 Linear cryptanalysis *

1.3.5 Weak keys *

2 Data Encryption Standard (DES) *

2.1 Description of DES *

2.1.1 Initial Permutation *

2.1.2 The Key Transformation *

2.1.3 The Expansion Permutation *

2.1.4 S-Box Substitution *

2.1.5 P-box Permutation *

2.1.6 Final Permutation. *

2.1.7 Decrypting DES *

2.2 Attacks on DES *

3 The Advanced Encryption Standard (AES) *

[3.1 Introduction *](#)

[3.2 Round 2 finalists *](#)

[3.2.1 MARS *](#)

[3.2.2 RC6 *](#)

[3.2.3 Rijndael *](#)

[3.2.4 Serpent *](#)

[3.2.5 Twofish *](#)

[3.3 Comparison of the Finalists In Hardware *](#)

[3.4 Areas of comparison *](#)

[3.4.1 Area *](#)

[3.4.2 Throughput *](#)

[3.4.3 Transistor Count *](#)

[3.4.4 Input/Outputs \(I/O\) Required *](#)

[3.4.5 Key Setup Time *](#)

[3.4.6 Algorithm Setup Time *](#)

[3.4.7 Time to Encrypt One Block *](#)

[3.4.8 Time to Decrypt One Block *](#)

[3.5 Comparison of algorithms *](#)

[3.5.1 Mars *](#)

[3.5.2 RC6 *](#)

[3.5.3 Twofish *](#)

[3.5.4 Serpant *](#)

[3.5.5 Rijndael *](#)

[3.6 Conclusion *](#)

[4 The GSM Encryption Algorithms A3 A5 and A8 *](#)

[4.1 A3, The MS Authentication Algorithm& A8, The Voice-Privacy Key Generation Algorithm *](#)

[4.1.1 COMP 128 *](#)

[4.2 Description of A5 Stream Cipher *](#)

[4.3 Attacks *](#)

[4.3.1 COMP128 *](#)

[4.3.2 A5 *](#)

[5 The Attacks I implemented *](#)

[5.1 Computaional Attack on COMP128 *](#)

[5.1.1 Conclusion on Comp128 Attack *](#)

[5.1.2 Over-the-air cloning *](#)

[5.2 Brute Force attack on A5, known plaintext. *](#)

[5.2.1 Generate the first byte then compare *](#)

[5.2.2 Reduce the output *](#)

[5.2.3 C optimisation *](#)

[5.2.4 Finding the best compiler *](#)

[5.2.5 Results and Conclusions *](#)

[5.2.6 Hardware base attack *](#)

[5.3 Simulation of the GSM Environment *](#)

[6 Conclusion *](#)

[6.1 Room for improvement *](#)

[7 References *](#)

[8 Glossary *](#)

[9 Acknowledgements *](#)

[10 Appendix A - Hardware Performance Figures for the AES Finalists. *](#)

[11 Appendix B - A implemantation of A5 in C. *](#)

[12 Appendix C - A implementation of COMP128 in C. *](#)

[13 Appendix D – A Verilog implementation of A5 *](#)

[14 Appendix E – C code for Brute force attack on A5 *](#)

1 INTRODUCTION

1. *Permutation and Substitution Boxes*

1. P-Boxes

A P-box simply reorders the bits to different locations; this is equivalent to wire crossing. Later in DES you will see Expansion and Compression P-Boxes.

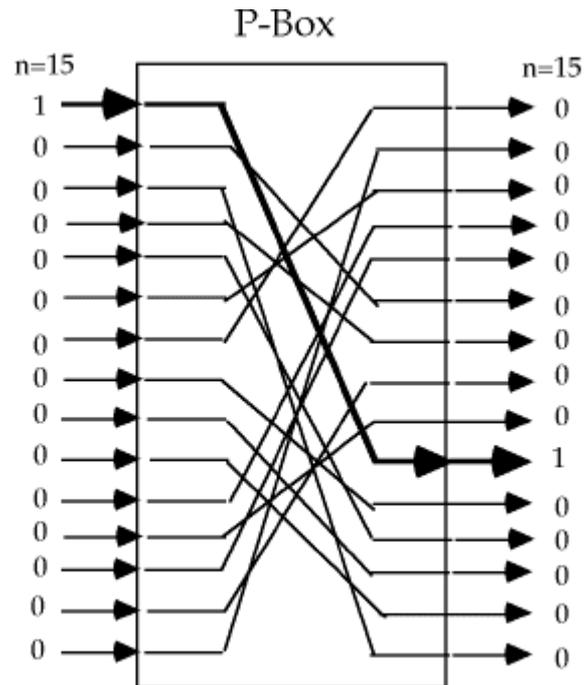


Fig 2.2 - Permutation or Transposition Function

Figure 1 P-Box

2. S-boxes

An $A \times B$ bit S-box replaces an A bit input with a B bit output.



Figure 2 S-Box

3. Substitution-Permutation Network

Also called a mixing transformation A Substitution-Permutation Network has sets of S-boxes linked by P-boxes.

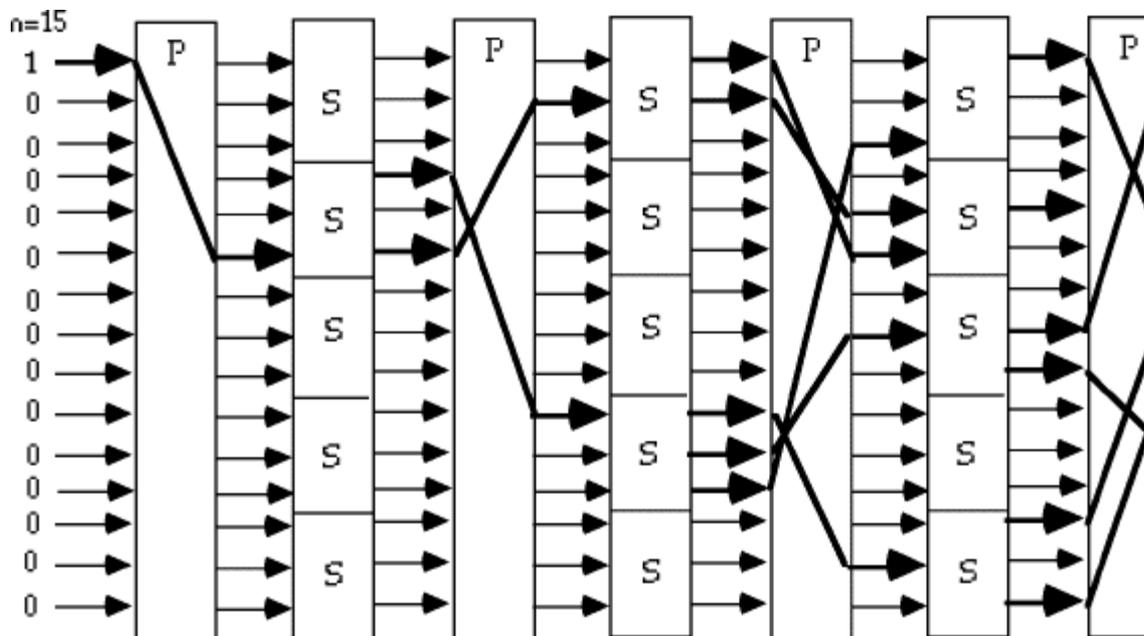


Fig 2.3 - Substitution-Permutation Network, with the Avalanche

Figure 3 Substitution-Permutation Networks, with the Avalanche Characteristic

2. Algorithms

1. Symmetric Algorithms

There are basically two types of key-based algorithms: symmetric and public-key. In symmetric algorithms the encryption key can be calculated from the decryption key easily (e.g. the reverse of the encryption key). So essentially with Symmetric Algorithms as long as the encrypted data is to remain a secret so must the key(s). There are two types of symmetric algorithm: Stream ciphers and Block ciphers. Both algorithms are used in GSM are symmetric ciphers, A5 is a symmetric stream cipher and A3 and A8 are example of a symmetric block cipher.

2. Block Ciphers

Block Ciphers operate on plaintext in groups of bits (blocks). A typical block size is 64 bits. With a block Cipher, the same plaintext block will always encrypt to the same ciphertext block, using the same key. DES is an example of a block cipher. DES encrypts 64-bit blocks of plaintext and products 64-bit of ciphertext. The key length is 56 bits the key can be any 56-bit number and can change at any time, all security rests in the key. DES applies a substitution followed by a permutation (this is known as a round) to each block 16 times.

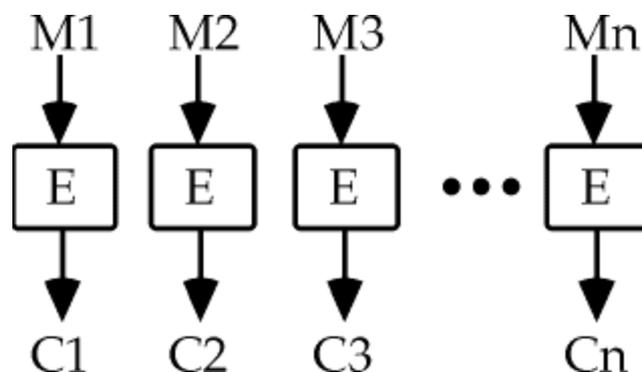


Figure 4 Block cipher ('M'= plaintext 'C' = ciphertext)

3. Public-Key Algorithms

Public Key encryption is not used in the GSM system, but for completeness I will briefly discuss it. Public-Key algorithms or asymmetric algorithms use two separate keys; a public key and a private key. The private key cannot be generated from the public key. The public key, as its name suggests, is available to everyone and is usually stored in a key database. It is used for encryption so anyone who wants to send you a message can download your public key, encrypt the message, and send it to you. On receipt of the ciphered message, you can decrypt it by use of your private key. All Public-Key algorithms are slower (approximately 1000 times slower [1]) than their symmetric counterparts.

4. Stream Ciphers

Stream Ciphers operate on streams of plaintext and ciphertext one bit or byte at a time. Using a stream cipher, the same plaintext bit or byte will encrypt to a different bit or byte every time it is encrypted. A stream cipher uses a Keystream Generator in conjunction with the plaintext. The Keystream Generator produces a "Random" string of bits, and these are used to generate a bit with each bit of the message, for example, each bit of the message could be XORed with a corresponding bit from the Keystream generator.

The strength of this system lies totally in the "randomness" of the Keystream that must be generated at both the Encrypting and Decrypting end of the communication channel. If it is totally random, then the Cipher is as perfectly secure as a one-time pad [1]. If it is, for example, a repeating set of 8 digits, it will be easily broken. It is vital that stream ciphers have session keys. The output of the Keystream generator is a function of this key. This means that if the Keystream for one communication is compromised, all other communications are still secure. Stream ciphers may be used to encrypt streams of communications traffic.

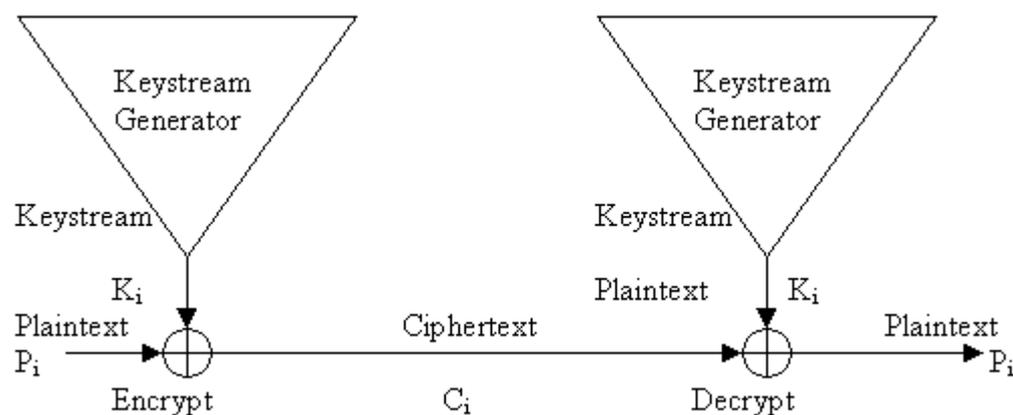


Figure 5 Stream cipher

3. *Cryptanalysis*

The objective of cryptanalysis is to "break the code", i.e. to recover the plaintext from the ciphertext, and preferably also to find which key was used.

1. Kerckhoff's Principle.

Kerckhoff's Principle_[16] is the general assumption that an opponent, "Oscar", knows which cryptosystem is being used. This is a reasonable assumption for the security reason. There are different levels of attacks on a cryptosystem.

Ciphertext-only attack: The assumption is the opponent possesses a string of ciphertexts, y .

Known plaintext. The opponent possesses a string of plaintexts, x , and the corresponding string of ciphertexts, y .

Chosen plaintext. The opponent has a temporary access to the encryption machinery, so that he can choose a plaintext string x and construct the corresponding ciphertext string y .

Chosen ciphertext. The opponent has a temporary access to the decryption machinery, so that he can choose a ciphertext string y and construct the corresponding plaintext string x .

2. Work Factor

Work Factor measures what is needed to carry out a specific analysis or attack against a cryptographic Algorithm. The attack is conducted under a given set of assumptions, which include the information available to achieve a predetermined

goal, such as the recovery of the plaintext or key [14]. Good Algorithms maximize the work need to compromise them.

In practice, there is no universally accepted, fixed set of parameters used to express the work factor. However, whatever it is measured in e.g. Hours, Number of Mathematical operations is usually converted into US Dollars.

3. Differential Cryptanalysis

Differential cryptanalysis is a type of attack that can be mounted on iterative block ciphers. S. Murphy first introduced this technique in an attack on FEAL-4 (Fast Data Encipherment Algorithm, 4 for rounds) [1] but this method was later improved and perfected by Biham and Shamir who used Differential Cryptanalysis to attack DES [30]. Differential cryptanalysis looks specifically at ciphertext pairs: pairs of ciphertext whose plaintexts have particular differences, but are encrypted under the same key. By careful analysis of the available data, probabilities can be assigned to each of the possible keys and eventually the most probable key is identified as the correct one.

Differential cryptanalysis has been used against a great many ciphers with varying degrees of success. In attacks against DES, its effectiveness is limited by carefully engineered S-boxes designed for DES in the mid-1970s. Differential cryptanalysis has also been useful in attacking other cryptographic algorithms such as hash functions.

No. of Rounds	Chosen Plaintext	Known Plaintext	Analyzed Plaintext	Complexity Analysis
8	2^{14}	2^{38}	4	2^9
9	2^{24}	2^{44}	2	2^{32}
10	2^{24}	2^{43}	2^{14}	2^{15}
11	2^{31}	2^{47}	2	2^{32}
12	2^{31}	2^{47}	2^{21}	2^{21}
13	2^{39}	2^{52}	2	2^{32}
14	2^{39}	2^{51}	2^{29}	2^{29}
15	2^{47}	2^{56}	2^7	2^{37}
16	2^{47}	2^{55}	2^{36}	2^{37}

Figure 6 Differential Cryptanalysis Attacks against DES [1]

4. Linear cryptanalysis

Matsui and Yamagishi first devised linear cryptanalysis also on an attack on FEAL [32]. It was extended by Matsui to attack DES [33]. Linear cryptanalysis is a known plaintext attack and uses a linear approximation to describe the behavior of the block cipher. Given sufficient pairs of plaintext and corresponding ciphertext, bits of information about the key can be obtained and increased amounts of data will usually give a higher probability of success.

There have been a variety of enhancements and improvements to the basic attack. Langford and Hellman introduced an attack called "*differential-linear cryptanalysis*", which combines elements of differential cryptanalysis with those of linear cryptanalysis [34]. Also, Kaliski and Robshaw showed that a linear cryptanalytic attack using multiple approximations might allow for a reduction in the amount of data required for a successful attack [35].

5. Weak keys

Weak keys are secret keys with a certain value for which the block cipher in question will exhibit certain regularities in encryption or a poor level of encryption. For instance, with DES, there are four keys for which encryption is exactly the same as decryption [1]. This means that if one were to encrypt twice with one of these weak keys, then the original plaintext would be recovered. For IDEA (International Data Encryption Algorithm), there is a class of keys for which cryptanalysis is greatly facilitated and the key can be recovered. However, in both these cases, the number of weak keys is such a small fraction of all possible keys that the chance of picking one at random is exceptionally slight. In such cases, they pose no significant threat to the security of the block cipher when used for encryption.

Of course, for other block ciphers, there might well be a large set of weak keys for which the chance of picking a weak key is more probable. In such a case, the presence of weak keys would greatly weaken the security of the cipher [40].

Weak key Value (with parity bits)	Actual Key
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 0E0E 0E0E	0000000 FFFFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFFF

Figure 7 DES Weak Keys [1]

Data Encryption Standard (DES)

1. Description of DES

DES has been a worldwide standard for 25 years. DES is an NSA (National Security Agency)-evaluated algorithm that was made public. Although it is showing signs of old age it is still reasonably secure. DES has been crypto analysed since its release so it is regarded as an excellent starting place when studying cryptology. DES is a symmetric cipher, specifically a 16-round Feistel cipher (see below) and was originally designed for implementation in hardware [1]. It is a block cipher that operates on 64-bit blocks. Its key length is 56-bits as its 64-bit key is reduced to 56-bits as every 8th bit is used for parity. Its structure is as follows.

1. Initial Permutation

This Permutation splits the 64 bit block into two 32-bit Blocks, it takes the 58th bit and puts it into the first left space, then the 50th bit and puts it in the second and then it takes the 42nd bit...and so on.

Left block; 58,50,42,34,26,18,10,2,60,52.....8th bit

Right Block; 57,49,41,33,25,17,9,1,59,51.....7th bit

This separation is to facilitate the Feistel structure of the cipher. In the classic Feistel structure, half of the data block is used to modify the other half of the data block, and then the halves are swapped.

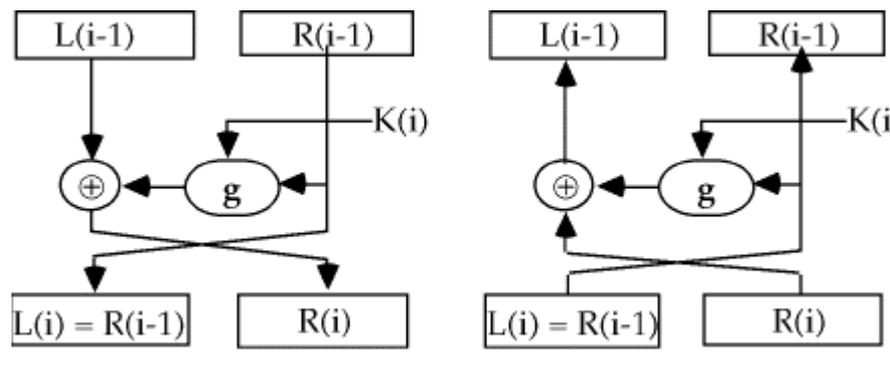


Figure 8 A round of a Feistel Cipher

2. The Key Transformation

Each round has an individual key of 48 bits. The individual keys are generated by dividing the 56-bit key it into two 28-bit parts. Then the bits are circularly shifted left one or two bits depending on the round. Then 48 out of the 56 bits are selected by a compression permutation [1]. As the key is shifted before each round, the 8 bits that are ignored are different for each round.

Rounds 1,2,9,16 are one shift and all the others are shifted left twice.

Round No.	Number of circular shifts
1,2,9,16	One
3,4,5,6,7,8,10,11,12,13,14,15	Two

Figure 9 Number of circular shifts for each round

3. The Expansion Permutation

This operation expands the right half of the plaintext from 32-bits to 48-bits so that it can be XORed with the round key. The 32 bits are split into 8 groups of four bits, where the first and last bit of the group of four appear twice in the new 48 bit number and the two middle bits only appear once. For example the 3rd bit of the 32-bit number would become the 4th bit, but the 4th would become both the 5th and 7th bit of the new 8 – bit number.

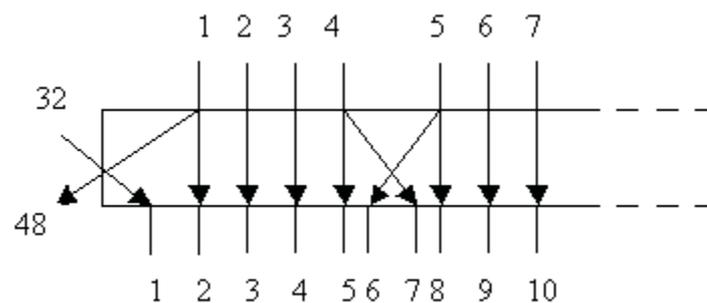


Figure 10 The Expansion Permutation Box in DES

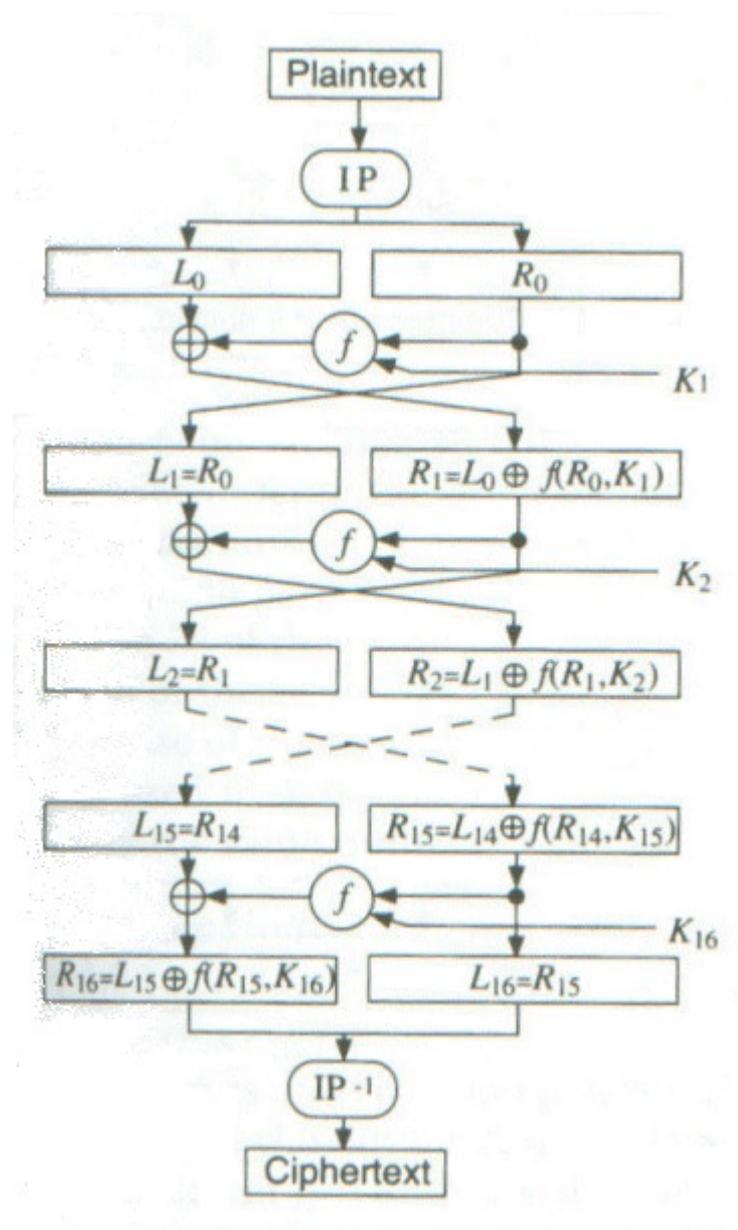


Figure 11 The structure of DES [1]

4. S-Box Substitution

After the round key is XORed with the expand plaintext block, the 48-bit result is divided into 8 6-bit sub blocks. Each block is operated on by a different s-block. The six-bit number that is inputted is used in a look-up table fashion to get a four-bit number.

The first and last bit gives the row x number and the other four bits give the column number y . Whatever 4-bit number is at row x and column y is output as part of the 32-bit number.

5. P-box Permutation

A straight permutation is performed. Each of the 32 bits is just reordered in a new location. For example bit 16 becomes bit 1 and bit 7 becomes bit 2 (see [1] For full

listing). Finally the result of the P-box permutation is XORed with the left half of the initial 64-bit block as per the Feistel filter structure. Then the left and right halves are switched and another round begins.

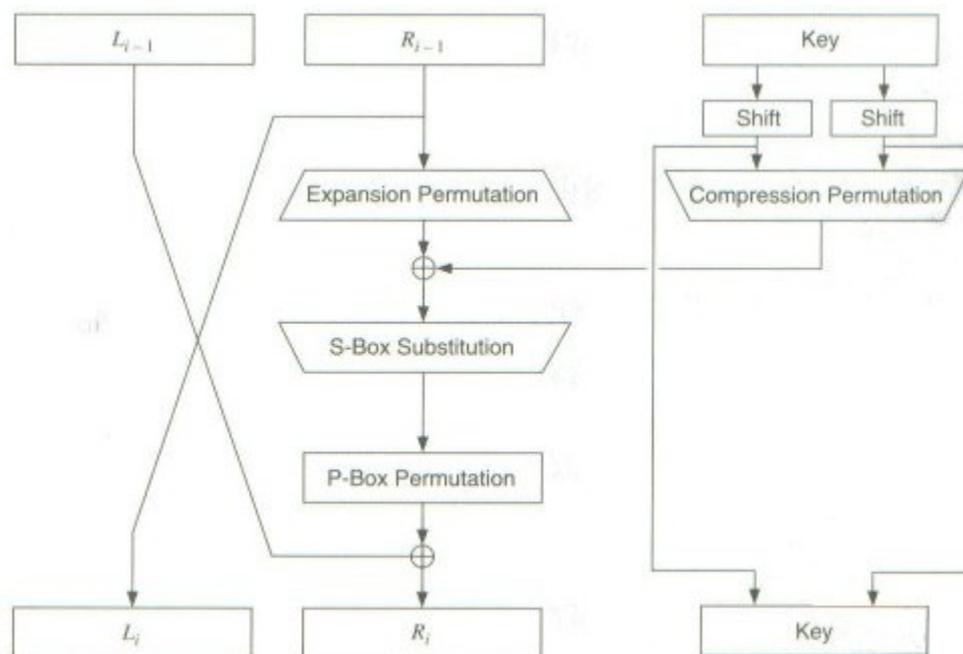


Figure 12 The structure of a round in DES [1]

6. Final Permutation.

The inverse of the initial permutation just reorders all the bits to the original places.

7. Decrypting DES

With DES it is possible to use the same function to encrypt or decrypt a block. The only difference is that the keys must be used in the reverse order.

1. Attacks on DES

No easy attack on DES has been discovered, despite the efforts of researchers for many years. The obvious method of attack is a brute-force exhaustive search of the key space; this process takes 2^{55} steps on average. In 1980 Hellman [37] showed a time-memory trade-off that allows improvement over exhaustive search if memory space is plentiful. There were also accusations the NSA had intentionally weakened DES. In 1998 Wiener estimated that a DES cracker could be built for one million dollars to give an average time of 35 minutes to break the algorithm [38].

The first attack on DES that was computational better than an exhaustive search was announced by Biham and Shamir using differential cryptanalysis [39]. This attack requires the

encryption of 2^{47} chosen plaintexts. Although it was a breakthrough, this attack is not practical because of both the large data requirements and the difficulty of mounting a chosen plaintext attack. Biham and Shamir have stated they consider DES secure.

More recently Matsui [41] has developed another attack, known as linear cryptanalysis (as mentioned in the previous chapter). By means of this method, a DES key can be recovered by the analysis of 2^{43} known plaintexts. The first experimental cryptanalysis of DES, based on Matsui's discovery, was successfully achieved in an attack requiring 50 days on 12 HP 9735 workstations.

On Tuesday, January 19, 1999, Distributed Net, a worldwide coalition of computer enthusiasts, worked with EFF's (The Electronic Frontier Foundation) DES Cracker and a worldwide network of nearly 100,000 PCs on the Internet, to win RSA Data Security's DES Challenge III in a record-breaking 22 hours and 15 minutes. EFF DES Cracker, which was built for less than \$250,000 tested 245 billion keys per second in conjunction with the computer network [42].



Figure 13 Paul Kocher, The DES Cracker and a "Deep Crack" chip

Paul Kocher, the machine's principal designer, displays one of the boards holding 64 custom search microchips. The six cabinets (behind) house 29 boards whose searching is coordinated by a PC (left). The machine, tests over 90 billion keys per second, taking an average of less than 5 days to discover a DES key. On the right one of over 1800 custom microchips, perform the DES key search. Each chip contains 24 search units that test possible keys against dual ciphertexts [42].

The consensus of the cryptographic community is that DES is not secure, simply because 56 bit keys are vulnerable to exhaustive search and not that there is a weakness within the algorithm. In fact, DES is no longer the standard for U.S. government use; triple-DES is in use

until AES (see Section 3) is ready for general use [36].

• The Advanced Encryption Standard (AES)

• *Introduction*

The Advanced Encryption Standard (AES) will be a new Federal Information Processing Standard (FIPS) Publication that will specify a cryptographic algorithm for use by U.S. Government organizations to protect sensitive (unclassified) information.

The AES is being developed to replace DES, but the NIST ([National Institute of Standards and Technology](#)) anticipates that Triple DES will remain the approved algorithm for the foreseeable future. Rijndael has been chosen as the AES algorithm. Rijndael's combination of security, performance, efficiency, ease of implementation and flexibility make it an appropriate selection for the AES [26].

Specifically, Rijndael appears to be consistently a very good performer in both hardware and software across a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its keysetup time is excellent, and its key agility is good. Rijndael's very low memory requirements make it very well suited for restricted-space environments, in which it also demonstrates excellent performance. Rijndael's operations are among the easiest to defend against power and timing attacks [26].

The following Chapter will detail both Rijndael and the four 2nd Round finalists. I will concentrate on Rijndael but also give a brief description of the other four algorithms.

The final section of this chapter is a short break down why Rijndael over the other finalists.

• *Round 2 finalists*

No.	Name	Designers
1.	Rijndael	Joan Daemen, Vincent Rijmen
2.	Mars	IBM
3.	RC6TM	RSA Laboratories
4.	Serpent	Ross Anderson, Eli Biham, Lars Knudsen
5.	Twofish	Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson

Figure 14 AES Round 2 Finalists

Below is a summary of each of the finalist in alphabetical order;

1. **MARS**

MARS is the IBM's entrant into the AES competition. As IBM is the company that designed DES, this alone makes this entry of particular interest.

The overall structure of MARS is as follows;

First, key material is XORed with the whole block (pre-whitening see glossary). Then, eight rounds of a transformation similar to DES are applied, but that transformation is fixed, and without any part that is affected by a key. These eight rounds of unkeyed transformation seem to be criticised by many sources [21 & 27], believing that they are wasteful and should not be used so early in the algorithm.

Then, there are sixteen rounds, which constitute the "cryptographic core" of the cipher. One 32-bit word is used to modify the other three, by being split into three copies of itself, and subjected to various manipulations, including one multiplication by key material, one S-box lookup, and two data-dependent rotations.

Then we have another unkeyed transformation, and another XOR of key material.

There are 40 32-bit words of subkeys, which are generated by a kind of shift-register method from the key.

The unkeyed rounds use two 8x32 bit S-boxes, addition, and the XOR operation. In addition to those elements, the keyed rounds use 32-bit key multiplication, data-dependent rotations, and key addition. Both the mixing and the core rounds are modified Feistel rounds in which one fourth of the data block is used to alter the other three fourths of the data block [22].

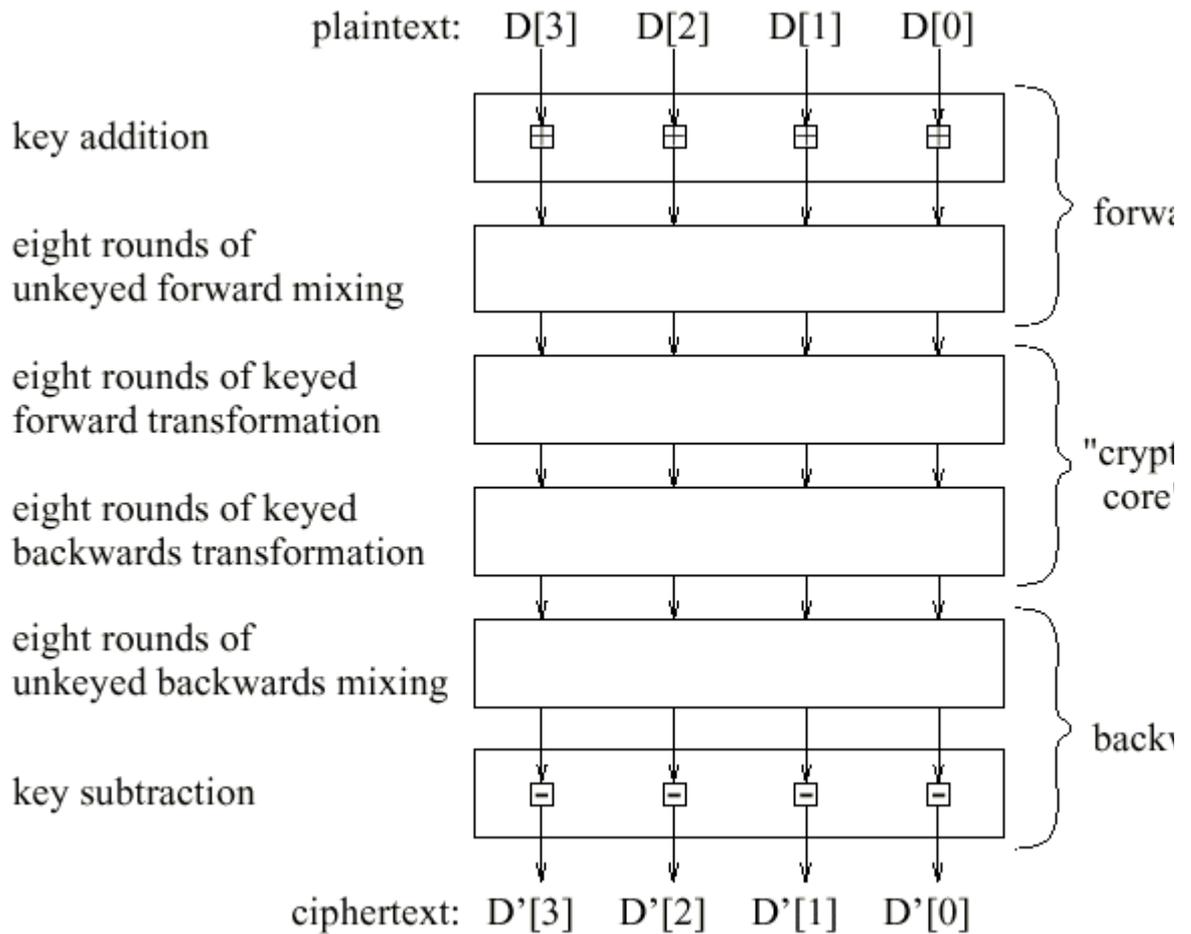


Figure 15 The structure of Mars

2. RC6

Devised by Dr. Ronald C. Rivest, RC6 is based on Feistel rounds; but not Feistel rounds operating between the two halves of the block. Instead, the Feistel rounds operate between pairs of quarters of the block, and they are interlocked by the exchange of some data. 20 rounds were specified for the AES submission.

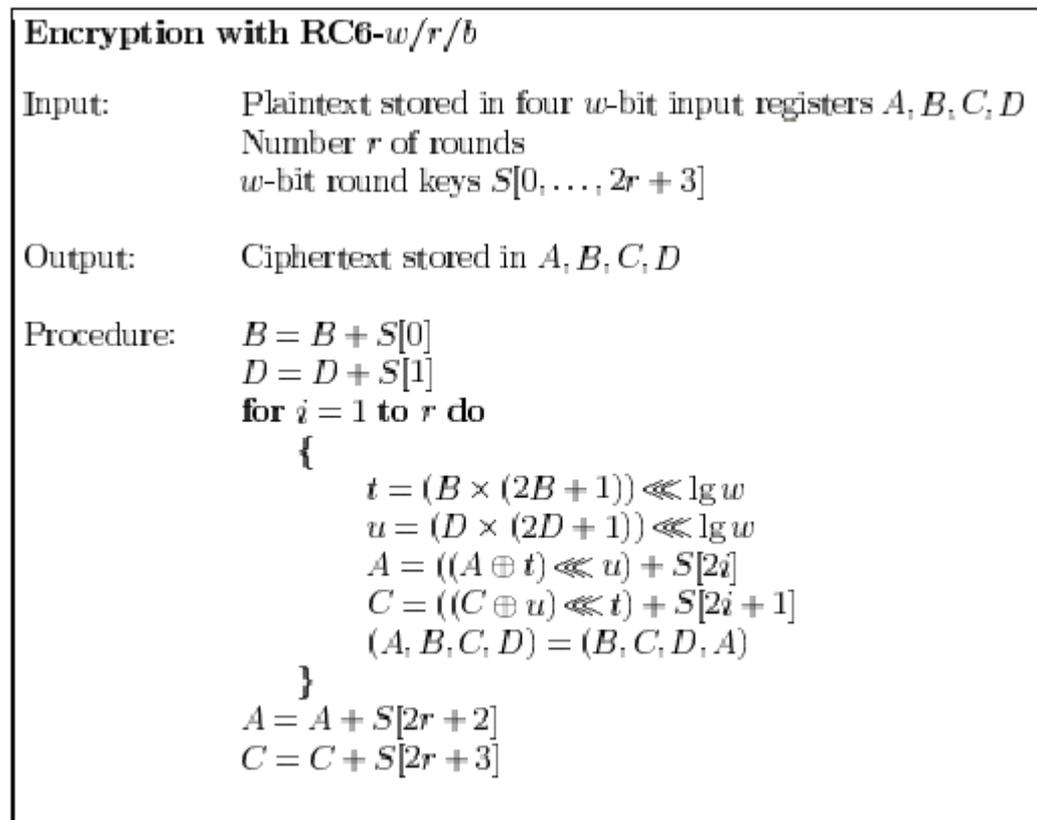


Figure 16 Summary of Encryption with RC6

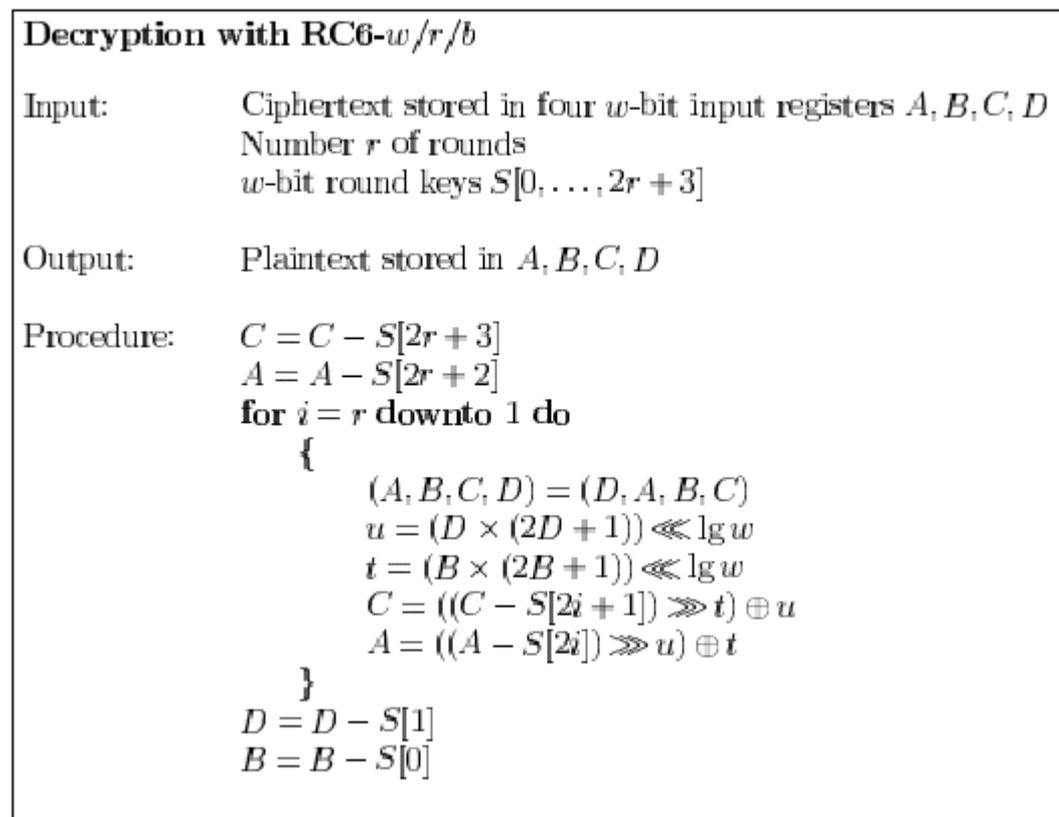


Figure 17 - Summary of Decryption with RC6

The round function of RC6 uses variable rotations that are regulated by a quadratic function of the data. Each round also includes 32-bit modular multiplication, addition, XOR, and key addition. Key addition is also used for pre- and post-whitening [24]. The design of the block cipher is such that the number of rounds, the size of the key, and the size of the block, are all flexible [27].

RC6 uses 44 subkeys, numbered S_0 to S_{43} , each one 32 bits long. The block to be enciphered is divided into four 32-bit integers, A, B, C, and D. The first four bytes enciphered form A, and the convention is little-endian; the first byte enciphered becomes the least significant byte of A. Each round of RC6 uses two subkeys; the first one uses S_2 and S_3 , and successive rounds use successive subkeys [24].

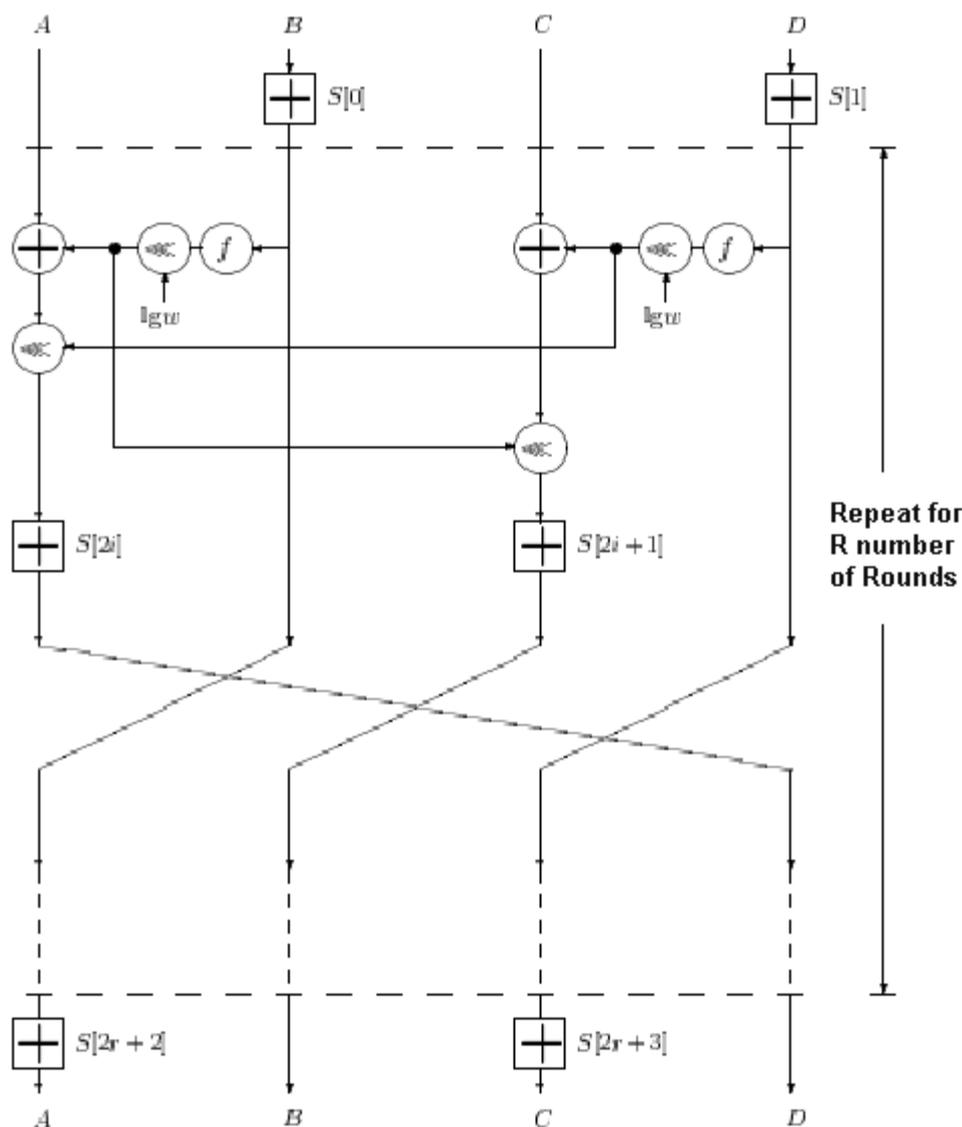


Figure 18 The structure of RC6

3. Rijndael

Rijndael is a substitution-linear transformation network designed to use only simple whole-byte operations with 10, 12 or 14 rounds, depending on the key size [17]. It provides extra flexibility over that required of an AES candidate, in that both the key size and the block size may be chosen to be any of 128, 192, or 256 bits. However, the variations of Rijndael which act on larger block sizes apparently will not be included in the actual standard [27].

Rijndael works as follows; A data block to be processed using Rijndael is partitioned into an array of bytes, and each of the cipher operations is byte-oriented. Rijndael's round function consists of four layers. In the first layer, an 8x8 S-box is applied to each byte. The second and third layers are linear mixing layers, in which the rows of the array are shifted, and the columns are mixed. In the fourth layer, subkey bytes are XORed into each byte of the array. In the last round, the column mixing is omitted [27].

1. The Rounds in detail

Each regular round involves four steps.

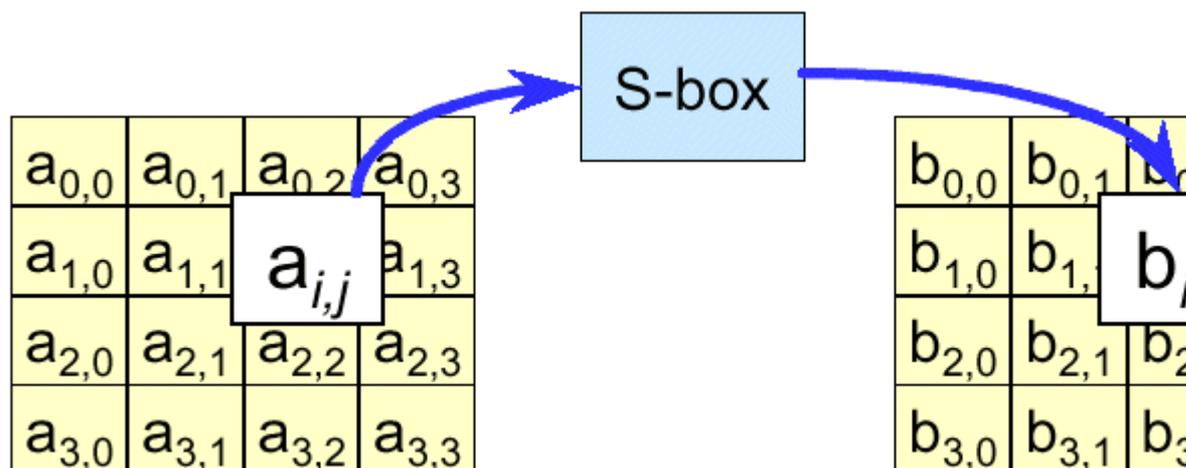


Figure 19 Step 1 of Round Byte-Sub [17]

First is the **Byte Sub** step, where each byte of the block is replaced by its substitute in an S-box.

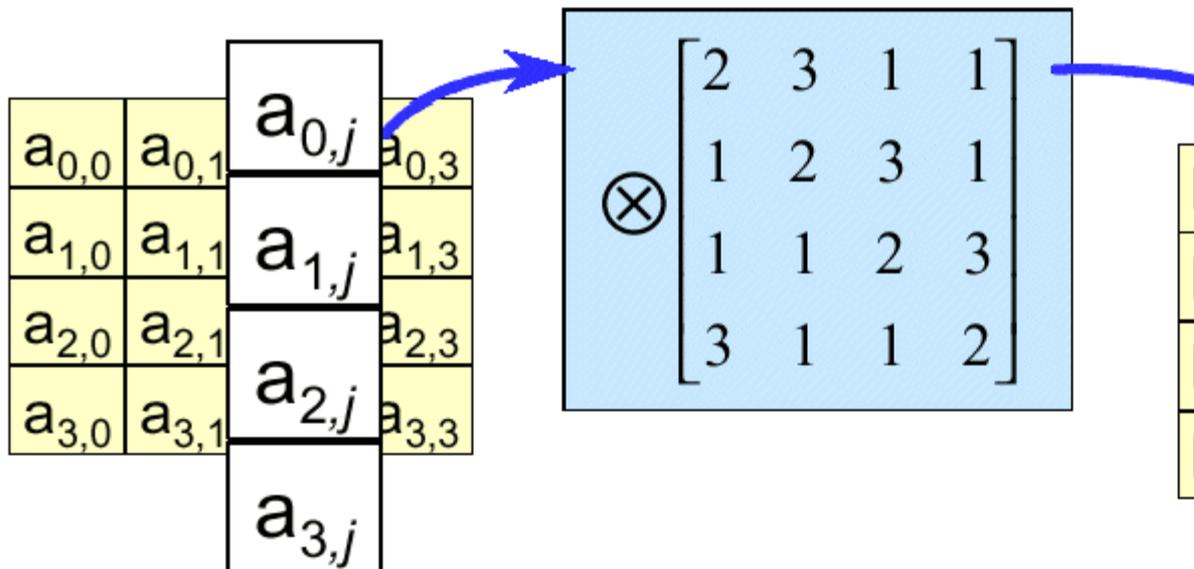


Figure 20 Step 2 of Round Mix Column [17]

Next comes the **Mix Column** step. Matrix multiplication is performed: each column, in the arrangement we have seen above, is multiplied by the following matrix:

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

Figure 21 The Multiplication matrix

This multiplication is done over GF(2⁸). This means that the bytes being multiplied are treated as polynomials rather than numbers. Thus, a byte "multiplied" by 3 is that byte XORed with that byte shifted one bit left. If the result has more than 8 bits, the extra bits are not simply discarded: instead, they're cancelled out by XORing the binary 9-bit string 100011011 with the result. This string stands for the *generating polynomial* of the particular version of GF(2⁸) used; a similar technique is used in cyclic redundancy checks.



Figure 22 Step 3 of Round Shift Row [17]

Considering the block to be made up of bytes 1 to 16, these bytes are arranged in a rectangle, and shifted as follows:

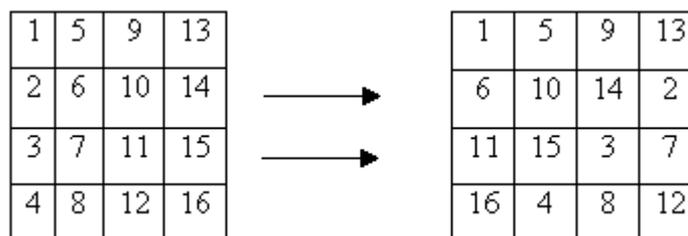


Figure 23 The blocks are shifted according to the above matrix [17]

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

 $+$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$

 $=$

Figure 24 Step 4 of the Round Key addition

The final step is **Add Round Key**. This simply Xor's in the subkey for the current round. The extra final round omits the Mix Column step, but is otherwise the same as a regular round. Because it begins and ends with an ARK (Add Round Key) step, there is no wasted unkeyed step at the beginning or end. Below is a graphical representation of the Rijndael Algorithm;

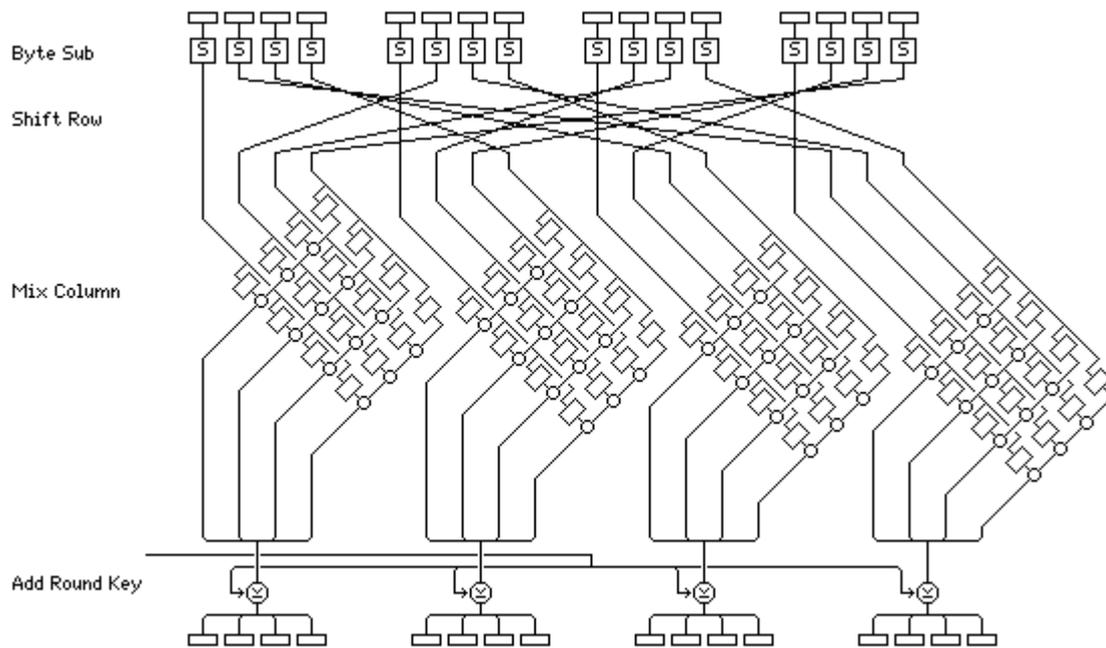


Figure 25 The structure of Rijndael

4. Serpent

Developed by Eli Biham, SERPENT is a "straight-through" algorithm (or substitution-linear transformation network), rather than a Feistel cipher. It consists of 32 rounds.

It is simple in appearance, using plain 4-bit-wide S-boxes without additional inputs and standard computer logic operations. It also includes an initial permutation and an inverse initial permutation, so that the S-boxes can be implemented with logic operations instead of table lookups; this is possible because the eight S-boxes used by the algorithm are used in sequence rather than in parallel [25].

The round function consists of three layers: the key XOR operation, 32 parallel applications of one of the eight specified 4x4 S-boxes, and a linear transformation. In the last round, a second layer of key XOR replaces the linear transformation.

In a normal round, the first step is to XOR the current round's subkey with the 128-bit-wide block (the key XOR operation). [27] Then, the entire block is transformed, nibble (4 bits) by nibble, according to the current S-box for the round. The S-boxes are numbered from S0 to S8, and are used in order repeatedly;

S0 for rounds 1, 9, 17, and 25,

S1 for rounds 2, 10, 18, and 26...

Then, the block goes through a series of mixing operations so that the different nibbles of the block interact.

This process proceeds, in bitslice mode, as follows; for the normal mode described here, this series of steps must be preceded by the inverse of the initial permutation and followed by the initial permutation to be correct. In the final round, the mixing operations are omitted. After the 32nd round, the bits are subjected to what is called in SERPENT the final permutation, which is the inverse of the initial permutation [25].

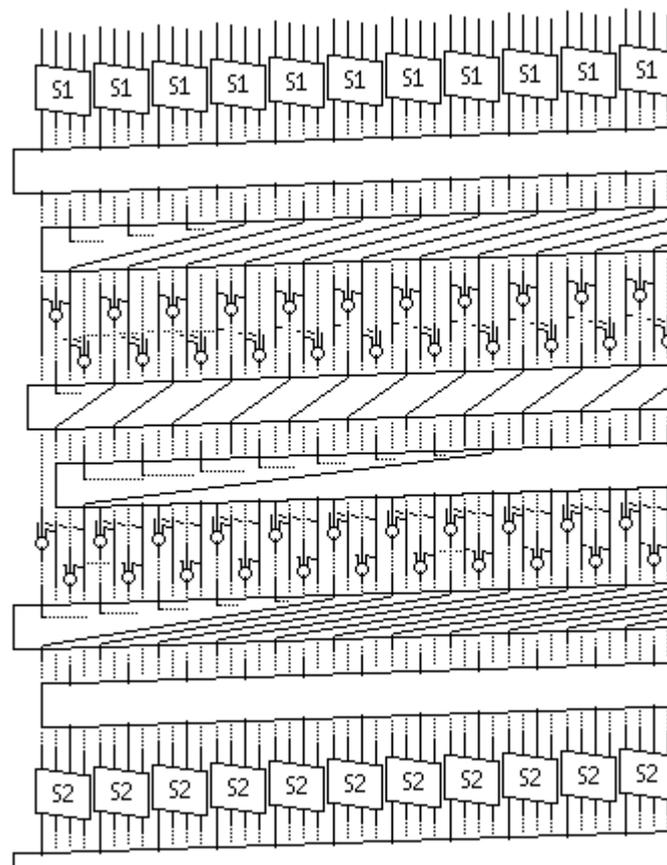


Figure 26 The structure of one round in Serpent

5. Twofish

Developed by Bruce Schneier as a successor to his 64-bit Blowfish block cipher. Schneier described it very succinctly as follows;

"Specifically, Twofish is a 128-bit block cipher that accepts a variable-length key up to 256 bits. The cipher is a 16-round Feistel network with a bijective F function made up of four key-dependent 8-by-8-bit S -boxes, a fixed 4-by-4 maximum distance separable matrix over $GF(28)$, a pseudo-Hadamard transform, bitwise rotations, and a carefully designed key schedule."

Twofish has a block size of 128 bits, and accepts a key of any length up to 256 bits. Twofish is fast on both 32-bit and 8-bit CPUs and in hardware. It may have in network applications where keys are changed frequently and in applications where there is little or no RAM and ROM available. [28]

It does not use a classic Feistel structure but rather a slightly modified version using 1-bit rotation. Twofish uses 40 32-bit subkeys. The first eight are used for whitening; four at the beginning and four

at the end are XORed with the entire block. Each round uses two of the remaining 32 subkeys, and so Twofish has sixteen rounds. The 128-bit block is divided into four 32-bit quarters. In Each round two of these 32 bit words is used as the input into the f function.

Each is broken up into Four Bytes. The four bytes are sent through four different key dependent 8x8 SiBoxes. The Four Output Bytes are combined into 32-bit words by use of what is known as a "Maximum Distance Separable Matrix". The 2 32 bit words are combined using the following;

Pseudo-Hadamard Transforms

A pseudo-Hadamard transform (PHT) is a simple mixing operation that runs quickly in software. Given two inputs, a and b , the 32-bit PHT is defined as:

$$a' = a + b \bmod 2^{32}$$
$$b' = a + 2b \bmod 2^{32}$$

This PHT can be executed in two opcodes on most modern microprocessors, including the Pentium family.

Figure 27 Pseudo-Hadamard Transform

Then they are added to the two round subkeys and finally Xored with the right hand side of the text.

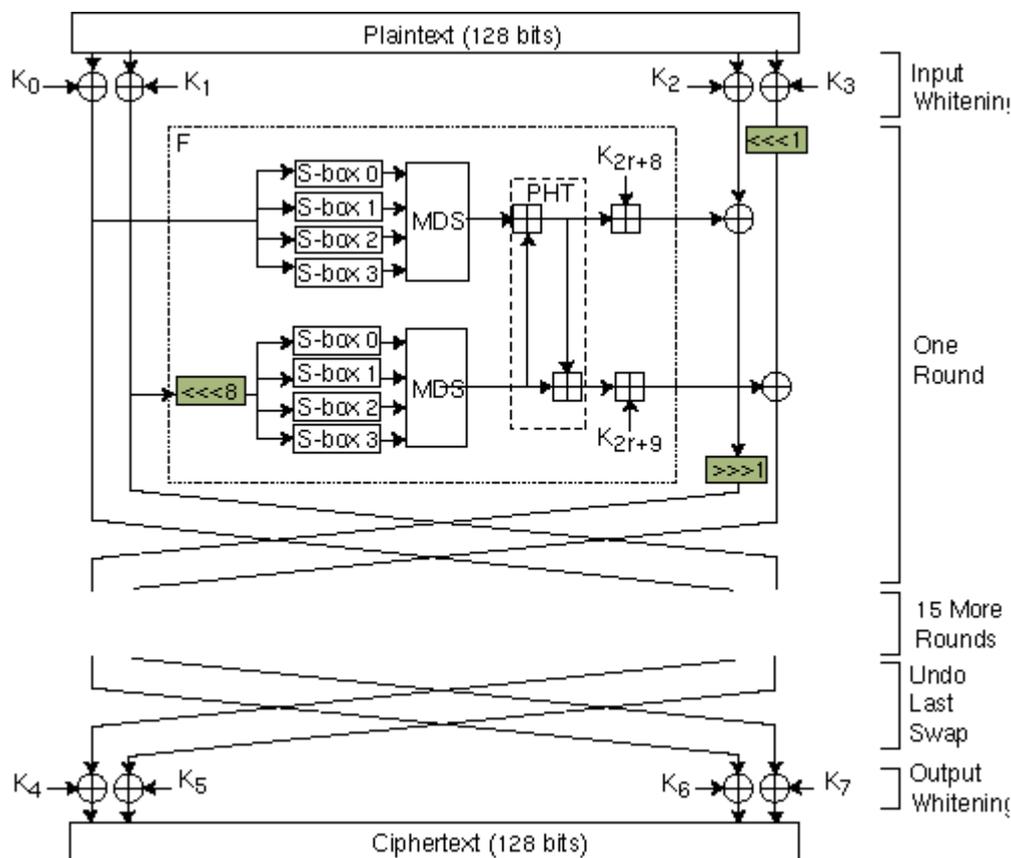


Figure 28 The structure of Twofish

• Comparison of the Finalists In Hardware

The NIST who were conducting the competition for the New Encryption standard enlisted The National Security Agency (NSA) to provide hardware performance measurements to aid NIST in their selection of the AES algorithm.

Round 2 consisted of tests to compare the hardware efficiency of the finalists. Round 1 was a mainly software based comparison. To cover a wide range of potential hardware applications, the NSA, used two distinct architectures, small area iterated version and a high speed, large area pipelined version. [29] The standard design approach consisted of creating hardware models using VHDL.

The NSA detailed all the findings in a document entitled Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms_[29]. This document is totally impartial and only gives fact about the algorithms and how they performed in the various hardware tests. I feel that this quote from the document sums up the NSA's impartiality;

"It should be emphasized that any data point based on a single is a relatively narrow view of the algorithm's overall performance or rating. For this reason, there was no attempt to rank algorithms

in order. Rather, it is left to the cryptographic community to establish a consensus of the most important parameters – in combination or alone – and to draw appropriate conclusions from the data provided herein." [29]

The Document is very comprehensive and testing the algorithms in a variety of ways;

- **Areas of comparison**

1. **Area**

The two varieties of architectures, iterative and pipeline will be on the extremes of area with the iterative being the smallest, and the pipelined being the largest.

2. **Throughput**

In most cases, throughput is directly proportional to area; as area decreases, throughput decreases. Therefore Iterative architecture has a much lower throughput than the pipeline version. Throughput is reported for both architectures.

3. **Transistor Count**

Transistor count is a more specific measure than area and is often more useful. The transistor count can be useful to estimating programmable logic implementations since these devices typically report the number of useable gates.

4. **Input/Outputs (I/O) Required**

With the goal of consistency among algorithms, the NSA fixed the I/O for all algorithms.

5. **Key Setup Time**

The key setup time refers to the amount of time needed before subkey expansion is ready to execute. Some algorithms use the user-supplied key directly in the subkey expansion thereby reducing the key setup time to zero. Others require some pre-calculation or translation of the key prior to subkey expansion steps.

6. **Algorithm Setup Time**

None of the evaluated algorithms contained an algorithm setup time.

7. **Time to Encrypt One Block**

This parameter will address minimum latency times for each of the algorithms. The time to encrypt one block, measured in nanoseconds, is a function of two parameters: the worst-case path delay between any two registers, and the number of rounds in the algorithm.

8. **Time to Decrypt One Block**

As above, this parameter will address minimum latency times for each of the algorithm submissions. Decryption does not always require identical processing as encryption. Therefore, the time required to decrypt one block is reported.

In *Appendix A* I have reproduced the summary of each of the implementations of the five final algorithms. There are two tables for each algorithm (exception Serpent) one for the standard 128 bit key size and a second table detailing the result of the 3 in 1 algorithm which is an implementation of the algorithm that can use 128,129 or 256 bit keys.

Below I have reproduced two tables giving short details on each of the algorithms summarizing the results and performance metrics. These comparison values are given only for the combined key size design, which implements a selectable 128-bit, 192-bit, or 256-bit key in the same device as mentioned above. But they detail the two hardware architectures, ie. Iterative and Pipeline.

Parameter	Algorithm			
	MARS	RIJNDAEL	RC6	SERPEN
Area (um2)	127,432,766	46,361,993	21,660,006	23,274,000
Transistor Count	1,950,277	1,029,046	430,436	345,400
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	56.7	443.2	103.8	202.0
Key Setup Time Encrypt (ns)	9553	0	8139	19,000
Key Setup Time Decrypt (ns)	27470	288.8	8139	672.0
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	2256.9	288.8	1233.2	632.0
Time to Decrypt One Block(ns)	2256.9	288.8	1233.2	632.0

Figure 29 Iterative Summary [29]

Parameter	Algorithm			
	MARS	RIJNDAEL	RC6	SERPENT
Area (um2)	1,333,099,627	471,996,329	554,268,739	438,561,500
Transistor Count	20,667,308	7,130,697	9,013,872	5,741,460
Input/Outputs Required	520	520	520	520
Throughput (Mbps)	2189	5163	2171	8030
Key Setup Time Encrypt (ns)	3718	0	3660	18,900
Key Setup Time Decrypt (ns)	3718	233.99	3660	212.50
Algorithm Setup Time (ns)	0	0	0	0
Time to Encrypt One Block (ns)	1988	248	1179	510.0
Time to Decrypt One Block(ns)	1988	248	1179	510.0

Figure 30 Pipeline Summary [29]

• Comparison of algorithms

The NSA report contains multiple discussions comparing the 5 algorithms under the above criteria for both Hardware architectures and for both the 3-in-1 implementations and the single 128 bit implementation. After the study of the graphs and text within it, it becomes apparent that Mars is

clearly the least efficient Algorithm. (See Appendix A)

1. **Mars**

Mars uses six times more area than RC6, Twofish and Serpant, for the iterative implementation and almost 3 times more for the Pipelined version. Its transistor count is also way above the average. Its throughput for the Pipeline implementation is competitive with Twofish and RC6, but is particularly poor for the Iterative hardware. It consistently has the worst times to encrypt and decrypt a block and its key setup time is extremely poor in respect to all the algorithms bar RC6.

2. **RC6**

I would Rank *RC6* as the second worst algorithm, it does perform very well in both architectures in regard to having the smallest area and it also has a very competitive transistor count. It becomes very noticeable especially in the iterative hardware that its keysetup times and the time taken to encrypt a block are well below par. If you compare its throughput in the Iterative version it is approximately $\frac{1}{4}$ of Rijndael score and the same is true in the Pipeline Architecture. Where Serpant has a throughput of 8030 and RC6's throughput is only 2171.

3. **Twofish**

Twofish is similar to RC6 in that it does particularly well in regard to area and transistor count. It also has on average the best keysetup times. It only loses out on throughput and time to encrypt/decrypt a block which are very closely linked in most algorithms.

4. **Serpant**

Serpant is a very strong algorithm. It was initially chosen as a back up algorithm until the idea of a backup algorithm was disregarded. It performs brilliantly in the pipeline hardware beating Rijndael in everything except time to encrypt/decrypt a block. Unfortunately for Serpant, Rijndael excels in the Iterative hardware, with excellent throughput, keysetup time and time taken to encrypt or decrypt a block. That said, Serpant uses half the area used by Rijndael and just over $\frac{1}{3}$ of the transistors.

5. **Rijndael**

Below I have reproduced the NIST's reasons for selecting Rijndael as the new AES standard;

"When considered together, Rijndael's combination of security, performance, efficiency, ease of implementation and flexibility make it an appropriate selection for the AES. Specifically, Rijndael appears to be consistently a very good performer in both hardware and software across a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its key setup time is excellent, and its key agility is good. Rijndael's very low memory requirements make it very well suited for restricted-space environments, in which it also demonstrates excellent performance. Rijndael's operations are among the easiest to defend against power and timing attacks.[26]"

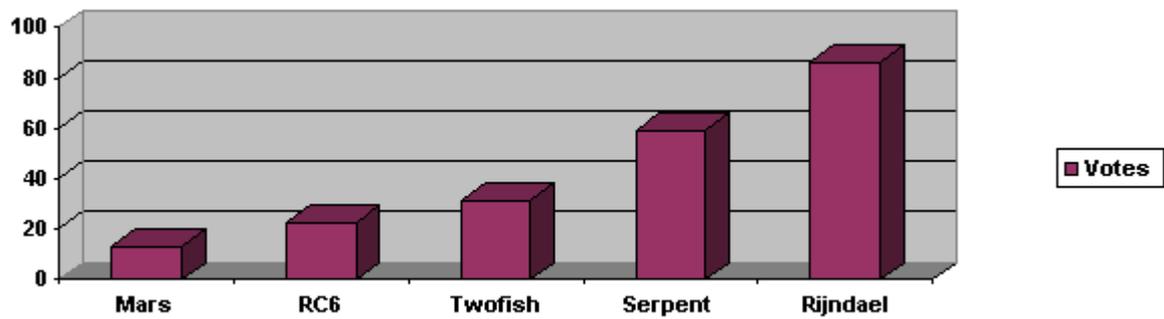


Figure 31 The votes for the round 2 finalists at the last AES conference

• Conclusion

The NIST have selected an extremely strong algorithm from five fine finalists, Serpant and Twofish being two excellent performers and equally worthy algorithms. To give an idea of the strength of Rijndael assume the "DES Cracker" mentioned in the previous chapter recovered a DES key in a second (i.e., try 2^{55} keys per second), then it would take that machine approximately 149 thousand-billion (149 trillion) years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old.^[27] it must also be remembered that the same implementation of Rijndael will also function with 192-bit and 256-bit keys. AES may last as long as DES, 20 years, assuming a weakness is not discovered in the algorithm. It should also be noted that it has been scrutinised for over a year by the cryptographic community.

• The GSM Encryption Algorithms A3 A5 and A8

1. A3, The MS Authentication Algorithm & A8, The Voice-Privacy Key Generation Algorithm

Authentication involves a challenge-response mechanism between two functional entities, the SIM in the Mobile Station (MS) and the Authentication Centre (AuC). When an account is initially generated for a new subscriber, a secret key (K_i), unique to that account is produced, one copy of which is stored in the SIM and the other in the AuC. When the MS notifies the MSC of its presence, the local VLR (Visitor Location Register) contacts the mobile unit's HLR (Home Location Register) and transmits the VLR's Location Area Identifier (LAI) and the mobile's IMSI. The HLR asks its local AuC for a set of triplets containing:

A 128 bit random number (RAND) for use as a challenge.

A 32 bit signed response (SRES) where $SRES = A_{03}(K_i, RAND)$ where K_i is the subscriber's key.

A 64 bit ciphering key K_c where $K_c = A_8(K_A, RAND)$.

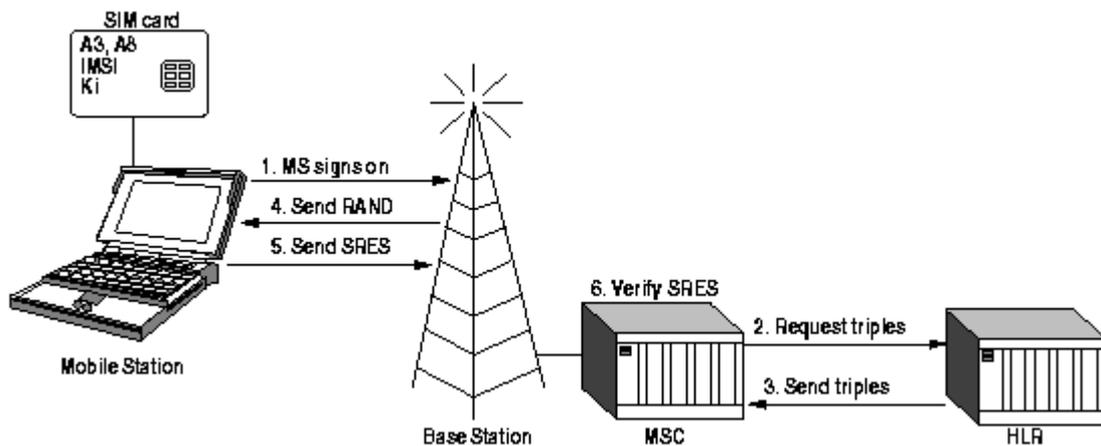


Figure 32 Mobile station authentication

The triplets are forwarded to the requesting VLR and each triplet is used only once for the authentication of the MS. The VLR transmits the challenge RAND to the mobile terminal, which uses the random number, in conjunction with the subscriber's secret key and the authentication algorithm A3 to generate SRES that is sent back to the VLR. If the received SRES is the same as that held by the VLR, the subscriber is authenticated. One session key, Kc, is used until the MSC decides to authenticate the MS again. This might take days [13]. A₃ and A₈ are both unpublished, key-dependant, one-way hash functions.

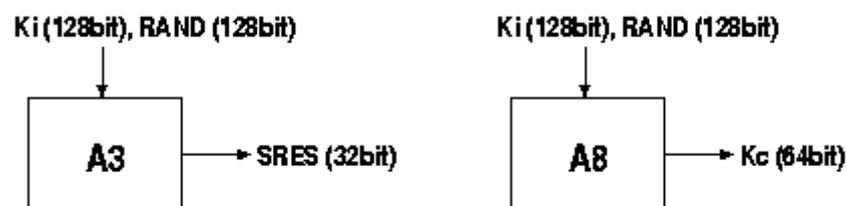


Figure 33 A3 & A8's Inputs and responses

1. COMP 128

Nearly every GSM operator in the world uses an algorithm called COMP128 for both A3 and A8 algorithms (As of April 1998 only five GSM Networks were known not to be using COMP128 [43]). COMP128 generates both the SRES response and the session key, Kc, on one run. It was given as an example algorithm that could be used in the GSM System Security Study leaked document [8].

I will briefly run through how Comp128 generates its output (I have reproduced the algorithm in C code in Appendix C as leaked from The GSM System Security Study and with the reverse-engineering fixes) .

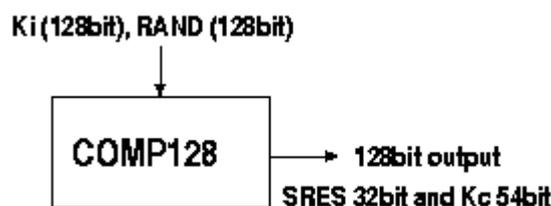


Figure 34 COMP128 calculation

Initially the RAND is loaded into a 32-byte register followed by Secret Key. So the first 128bits are the key and the rest is RAND. It consists of five P-Boxes. The 1st Box consists of 512 entries, the 2nd 256, the 3rd 128, the 4th 64 and final the 5th consists of 32 entries. The Algorithm consists of 5 rounds and during each round the register is altered 8 times by a selection of the above S-boxes. The algorithm therefore consists of 40 substitutions. This is followed by 128 permutations so every bit is transposed to a new location.

The first four bytes of the register are sent as the SRES. According to the leaked GSM document "simoutput[4..11] is Kc" which is 8 bytes which would be a 64 bit Key. On inspection of the C code (Appendix C) you can see that that Kc is bits [74..127] of the COMP128 output, followed by 10 zeros. In other words, A5 is keyed with only a 54 bits key. This represents a deliberate weakening of the key used for voice privacy by a factor of over 1000.

2. Description of A5 Stream Cipher

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_[6]. This version was later verified by the GSM organisation_[5]. I will be describing A5/1, which is used in the European market. AS A5/2 the American variant has been deemed totally insecure having been crack in 2^{16} iterations_[10].

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. A new session key Kc can encrypt each conversation. For each frame, Kc, 64-bit is mixed with a publicly known frame counter Fn, 22-bit and the result serves as the initial state of a generator, which produces 228 pseudo random bits. These bits are XORed 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 labeled 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 35).

When a register is clocked, its taps are XORed together, and the result is stored in the

rightmost bit of the left-shifted register, bit zero. The three registers are maximum 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 R_1 , bit 10 for R_2 , and bit 10 for R_3); 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.

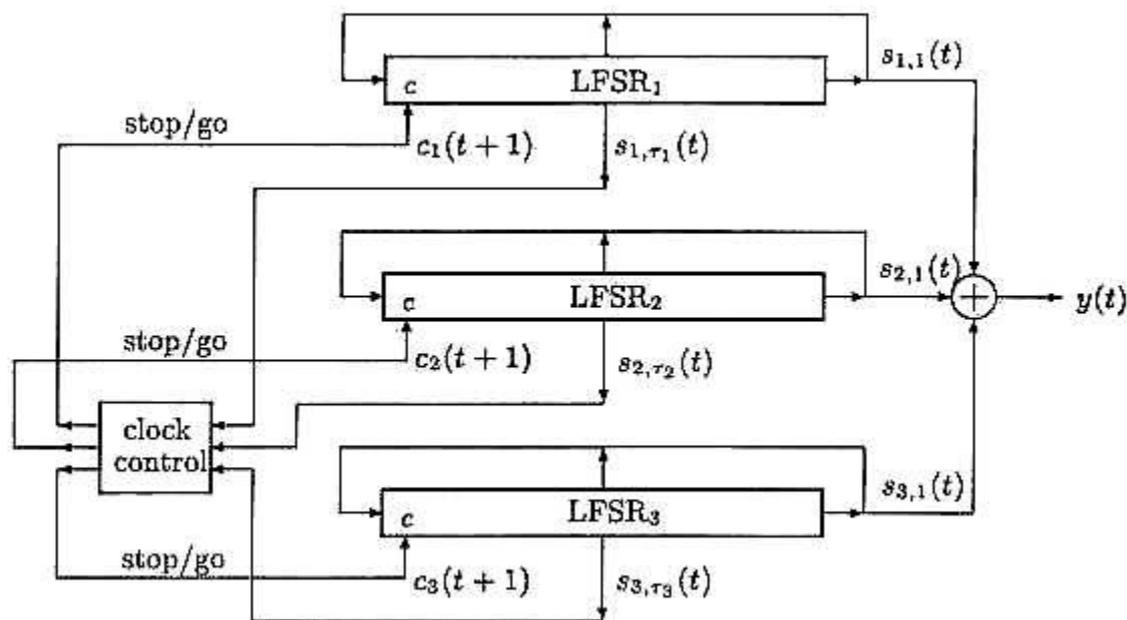


Figure 35 A5 keystream generator

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 XORed 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 are 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

See Appendix B for a C implementation of A5 as verified by the GSM organization.

3. Attacks

1. COMP128

In April 1988 Ian Goldberg and [Marc Briceno](#) [44] showed that COMP128 is cryptographically weak, and it is not difficult to break the algorithm and clone GSM digital phones. They formed a number of specially chosen challenges and query the SIM for each one. By analyzing they were able to determine the 128bit value of the Ki. This could then be used to clone another mobile and the billing would be attributive to the stolen phone.

To mount the attack you need physical access to the target SIM, an off-the-shelf smartcard reader, and a computer to direct the operation. The attack requires one to query the smartcard about 150,000 times; smartcard readers can issue 6.25 queries per second, so the whole attack takes 8 hours. Very little extra computation is required to analyze the responses.

The attack exploits a lack of diffusion: there's a narrow "pipe" inside COMP128. In particular, bytes $i, i+8, i+16, i+24$ at the output of the second round depend only on bytes $i, i+8, i+16, i+24$ of the input to COMP128. Bytes $i, i+8$ of the COMP128 input are bytes $i, i+8$ of the key, and bytes $i+16, i+24$ of the COMP128 input are bytes $i, i+8$ of the challenge input.

Now we "probe" the narrow pipe, by varying bytes $i+16, i+24$ of the COMP128 input and holding the rest of the COMP128 input constant. Since the rounds are non-bijective, you can hope for a collision in bytes $i, i+8, i+16, i+24$ of the output after two rounds. The birthday paradox guarantees that collisions will occur pretty rapidly (since the pipe is only 4 bytes wide); collisions in the narrow pipe can be recognized, since they will cause a collision in the output of COMP128 (i.e. the two authentication responses will be the same); and each collision can be used to learn the two key bytes $i, i+8$ with a bit of analysis of the first two rounds (i.e. perform a "2-R attack", in the terminology of differential cryptanalysis).

As stated, this would require $2^{\{4 \cdot 7/2 + 0.5\}} = 2^{\{14.5\}}$ chosen-input queries to COMP128 to learn two key bytes (since each of the four bytes of output after the second round are actually only 7-bit values), and thus would require $8 * 2^{\{14.5\}} = 2^{\{17.5\}}$ queries to recover the whole 128-bit key Ki. However, the authors used optimization to get this figure down.

This attack is not particularly novel a lot of research has been done in this area [46] ie. cryptographic hash functions of a FFT-like structure. COMP128 is a hash function of this design.

An Organization Called Echelon provides all the information necessary to clone a GSM phone, including software and hardware providers, <http://www.echelon.cx>.

COMP128 authentication works in other countries as well, because the local network asks the HLR of the subscriber's home network for the triplet (RAND, SRES, Kc). Thus, the local network does not have to know anything about the algorithms used. With 80 million GSM users, fixing flaws in such a widely fielded system is likely to be quite costly. A new authentication algorithm would have to be selected. Then new SIMs would have to be programmed with the new algorithm, and distributed to the 80 million end users. Finally, a software upgrade may be required for all authentication centres.

2. A5

1. *The Berkeley group*

The Berkeley group published and analysed A5/2 in August 1999. As the weaker of the two voice-encryption algorithms, it proved to be very weak. It can be broken in real-time without any trouble; the work factor is only 2^{16} for a 64 bit key algorithm [44]. Its weakness has been blamed on the NSA's hand in its design. They introduced a 4th LSFR, which decide on the clocking of the other 3 registers. This register is only 16 bits long hence the weakness of the algorithm.

2. *Briceno*

Briceno [6] found that in all the deployed versions of the A5/1 algorithm, the 10 least significant bits of the 64 key bits were always set to zero. Thereby reducing the complexity of exhaustive search from 2^{64} to 2^{54} . This flaw is essentially the fault of the implementation of the COMP128 algorithm see comment in Appendix C for more Details.

3. *Anderson and Roe*

Anderson and Roe [10] proposed a divide-and-conquer attack based on a known-plain-text attack. The attacker guessing the 41 bits in the shorter two registers and tries to determine the initial states of the 23 bits longer R_3 register from known keystream sequences. The attacker needs to know 64 successive keystream bits that can be retrieved if the attacker knows some cipher text and the corresponding plain text [12]. This depends largely on the format of the GSM frames sent back and forth. The GSM frames contain a lot of constant information, e.g. frame headers. The required 64 bits might not always be known, but 32 to 48 bits are usually known, sometimes even more keep in mind that the attacker needs only one 64-bit plain text segment. This would be a 2^{40} attack, if the clocking of the first two registers were not dependent on the third register. Because the middle bit of the third register is used for clocking, we have to guess about half of the bits in the third register between the clock bit and the LSB as well. This fact increases the time complexity from 2^{40} to 2^{45} [10].

4. *J. Golic*

J. Golic [12] has proposed another divide-and-conquer attack based on the same assumptions with the average complexity of $2^{40.16}$. Golic showed that only $2^{62.32}$ internal states could be reached from the 2^{64} initial states [12]. Based on this assumption, he describes how to obtain linear equations by guessing n bits in the LSFRs. By solving these linear equations, one could recover the initial states of the three LSFRs. The complexity of solving the linear equations is $2^{41.16}$. On average, one would resolve the internal state with 50 per cent chance in $2^{40.16}$ operations. However, each operation in this attack is much more complicated than the Anderson and Roe attack, as 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, but would be faster in Hardware [5].

Golic also proposed a Time-Memory Trade-Off Attack based on the Birthday Paradox (see glossary) in the same paper [12]. The idea behind this attack is to recover the internal states of the three LFSRs at a known time for a known keystream sequence corresponding to a known frame number, thus reconstructing the session key, K_c . He 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} 128bit 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 $O(2^{28})$. It takes Approximately 6 milliseconds to Access the fastest commercially available PC hard drive. This would mean that it would take almost 3 weeks to do the necessary probes [5]. 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.

5. *Biryukov, Shamir, Wagner*

Biryukov, Shamir, Wagner detailed two attacks in a paper titled "Real Time Cryptanalysis of A5/1 on a PC" [5] and several variants. They describe a Biased Birthday attack, which requires 2^{42} preprocessing steps and 292 GB of Hard drive space, yet with two minutes of data can extract the key in 2 second. A variant requires 2^{48} Preprocessing steps and 146 GB of Disk space where the conversation need only be 2 second long and can generate the key in several minutes.

The idea behind the Bias Birthday attack is to search through each of the frames of the conversation for 2 minutes. They are looking for an occurrence of a predefined string of 16 bits alpha (e.g. 10000...0) from bits number 101 to 277. During the first 2 minutes of the conversation approximately $(2^{-16} * 177 * 120 * 1000 / 4.6)$ 71 states containing alpha should occur they are called "green-states". Stored on the hard drives are chosen initial states called "red states". A state is coloured red if its sequence of output bits contain alpha between steps 101 and 277. The state of A5 contains 64 bits, but they keep only special states and thus we can encode them efficiently with shorter 48 bit names. From a red state there is little computation to product all of it's green states, and there is also very little computation in generating all of the possible keys that formed this red state (Red-states are the states after the 100 mixing clocks).

When a green state is found in the conversation, you work out where alpha occurred in the green state and then probe the hard drive to find the red state that would product alpha at this time. You then generate all of that red state's green states and if one is the same as the one in the conversation you generate all that red states possible key. Then you test all the key to see if one is correct and if so you have the session key. If not you wait for the next occurrence of alpha.

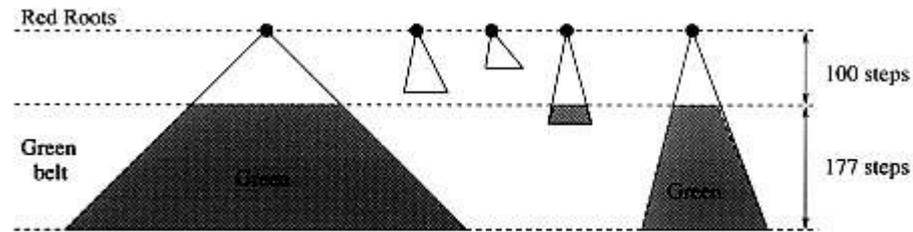


Figure 4: Trees of different sizes.

Figure 36 Trees of different sizes

The crucial point about the bias birthday attack is that in A5 there is a huge variance in the weight of the various red states. They found that the weight of about 85% of the states was zero, because their trees died out before reaching depth 100. So they optimised the attack in the pre-processing stage by only storing the heaviest tree. Each stored state increases the coverage by 12,500 green states on average, which improved their attack by almost two orders of magnitude.

The Random subgraph attack is very similar only after getting the first occurrence of alpha which usually occurs in the first 2 seconds they try to find the session key on this state alone by storing twice as many red states.

• The Attacks I implemented

1. *Computational Attack on COMP128*

After reading extensively about the weakness of COMP128 I decided to attempt to break the Algorithm and compromise the Secret key K_i . Smart cards have the following structure;

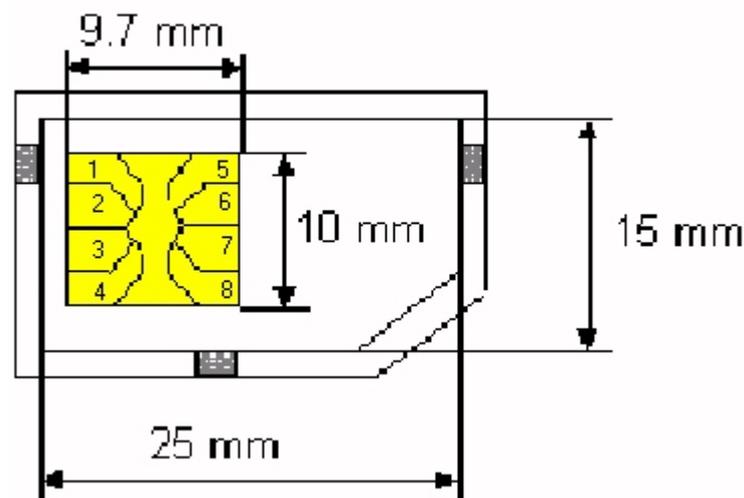


Figure 37 Structure of a Sim card

A smart card is a plastic card, with an embedded computer chip, usually an 8-bit

microprocessor, RAM (1/4 of a kilobyte), ROM (6 to 8 kilobytes), and either EPROM or EEPROM (a few kilobytes). The card has it's own operating system programs and data, and gets it's power from the reader.

A communication between the outside world and the card involves the following steps:

1. Activation of the contacts by the smart card reader
2. Resetting of the card by the reader
3. Answer-to-reset by the card
4. Optional selection of a protocol type
5. Processing of successive commands
6. Deactivation of the contacts by the card reader.

C1: Power supply (VCC)	C5: Ground (GND)
C2: Reset (RST)	C6: Programming voltage (VPP)
C3: Clock (CLK)	C7: Input/output (I/O)
C4: Reserved (RFU)	C8: Reserved (RFU)

Figure 38 The contacts and their uses on a Smart card

Input/output involves asynchronous characters transmitted in half-duplex mode. Each character is ten consecutive bits: a start bit, eight data bits, and an even parity bit. A short interval or "guard time" between successive characters allows for synchronization in the transmission.

I built the following circuit, to implement a smart card reader,.

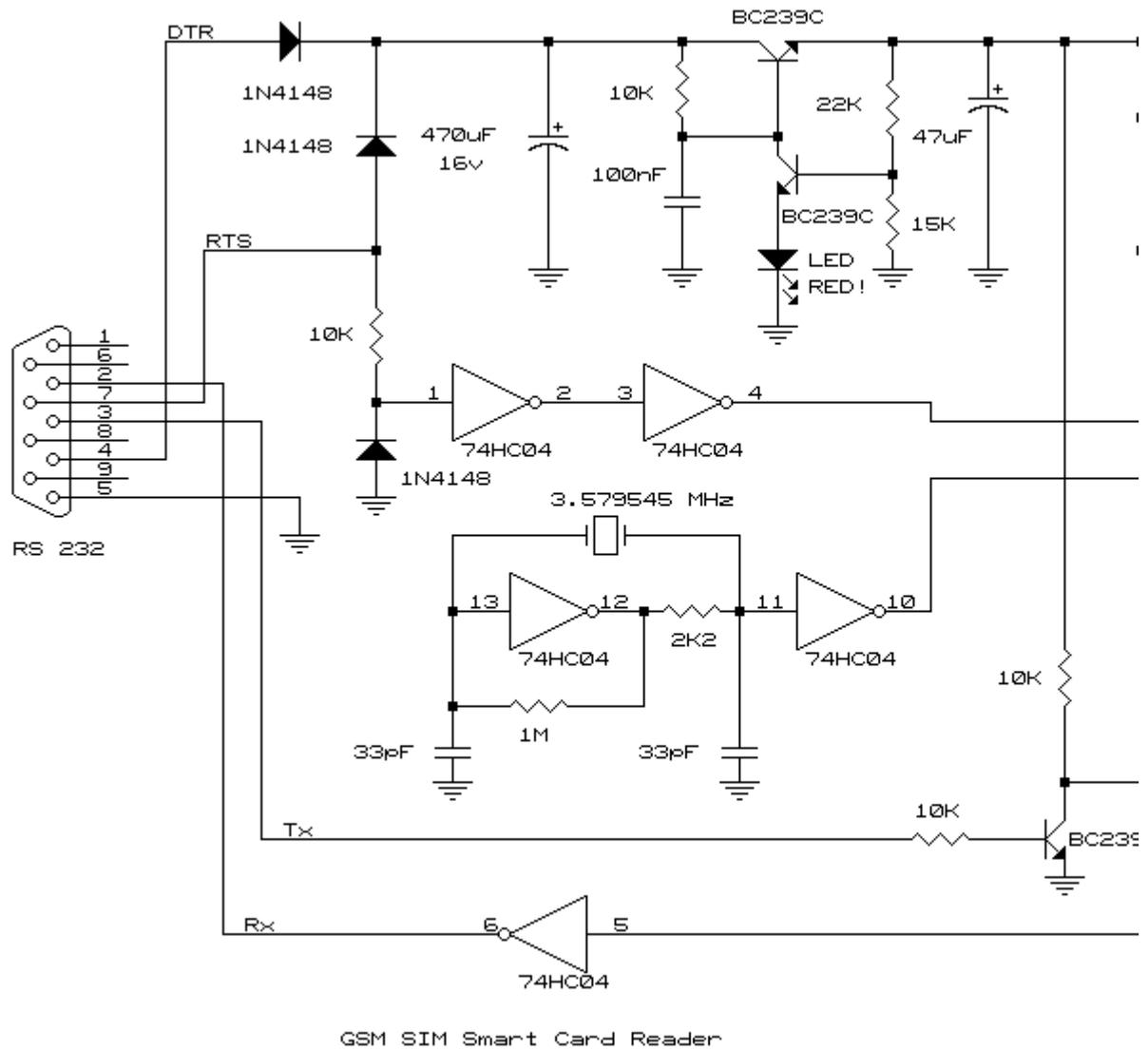


Figure 39 The Circuit for a Smart Card Reader.^[3]

The Program that this circuit is used with is called "Sim Scan" which was designed by Dejan Kajevic.^[3] It allows functional analysis of your GSM SIM smart card. The circuit has no external power source it obtains all of its power from the RS232 cable (Serial Port). It was designed to read ISO 7816 smart cards, GSM smart cards are of this type ^[3]. The Program lets you analyse the following on any smart card.

ATR: "Answer To Reset" strings.

CLA: Application class

INS: Instruction code

Files: Any files that are stored on the smart card

Ki: The scerect Key for the COMP128, only applicable to GSM SIM cards.

Some Smart cards may be destroyed buy using the Ki search (e.g. PrePaid cards) as they only have a limited number of runs of the A3A8 algorithm usually 10000 to 65536 times [2].

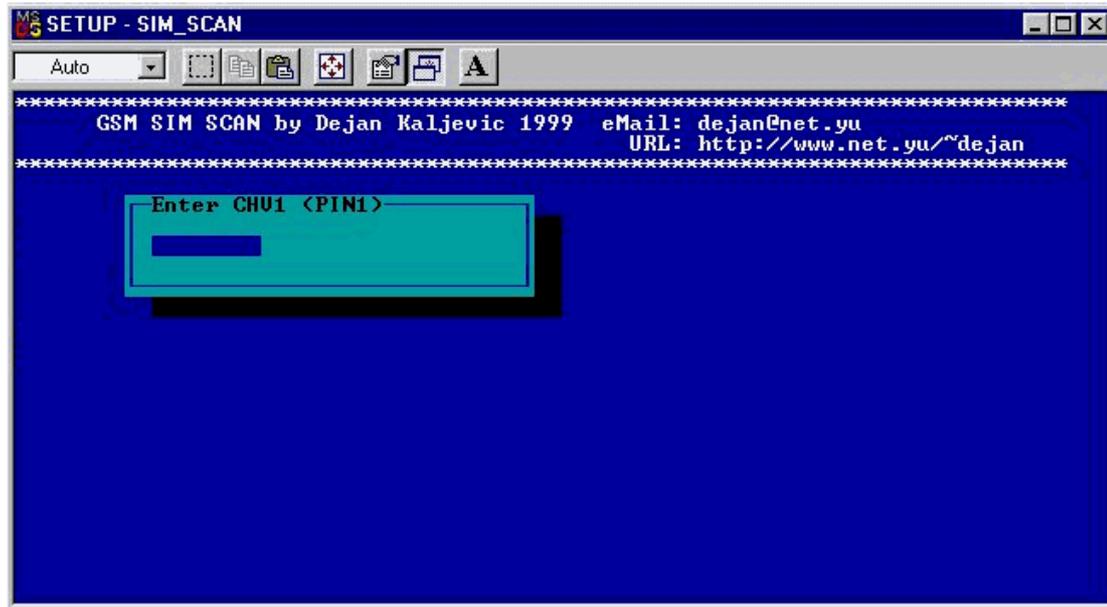


Figure 40 Sim Scan asking for the Pin of the card to be cracked

This version of the COMP128 attack implemented in the program is only a simple method that allows key finding in 90% of cases. It is similar but not identical to the method mentioned above. During the work it is possible to interrupt the program by pressing any key. In case of interrupting, a temp file is saved, and later you may continue the analysis from that point.

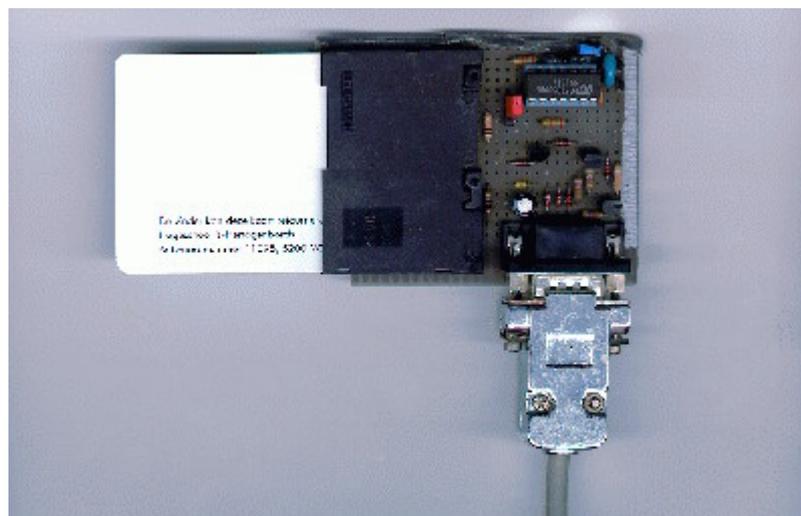
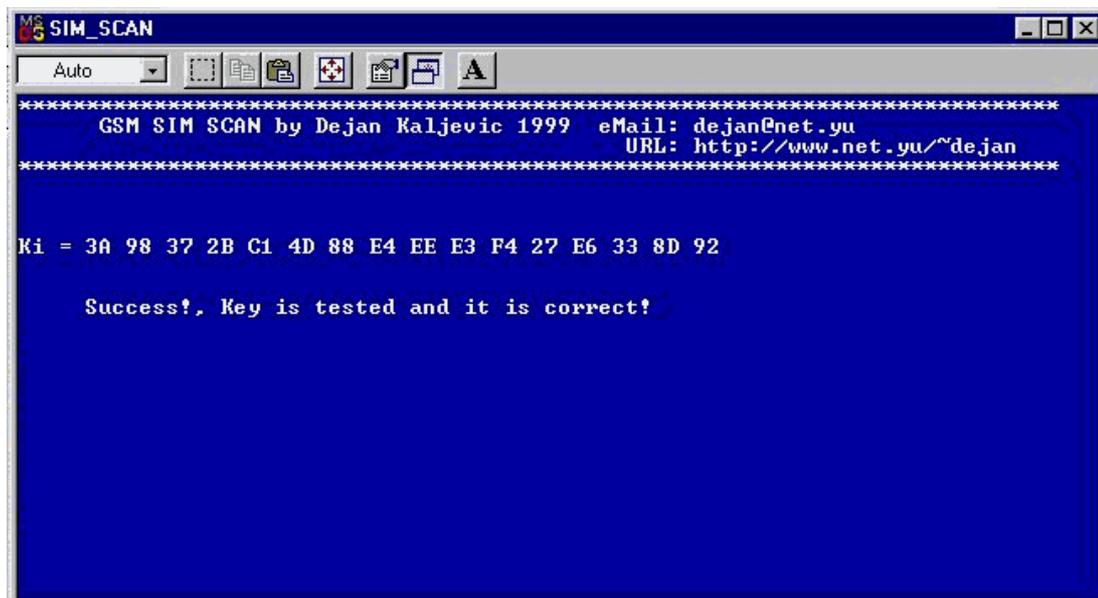


Figure 41 Smart Card Reader attached to serial port.

Most of the components were readily available from the college stores. The sim card holder and the low powered hex inverter (74HC04) were not available so I sent to Farnell for these and some other components. During Testing I discovered several mistakes and rectified them. But after building the circuit to the exact specification described above I was still unable to get

the circuit to operate. After further testing It seemed that I was not getting enough power to power either the sim card or the hexs inverter both of which needed 5volts. It seem that the designer had mis calculated the voltage drop over the transistors. After using external power to power these two component I was able to get the simcard reader operational.

I obtained an old prepaid sim card and began an attack on the card. Initially the program seemed very unstable and frequently halted because it was not receiving an "answer-to-reset" from the card. When this happened I would have to restart the program fortunately because of the temp file I was able to start from where the program had crashed. After consulting with the technicians in the lab I got the data sheets for some of the components and discovered that the crystal oscillator in the circuit could vary by + or - 20% of it's printed value. After trying several crystal one seemed to match the baud rate and made the circuit much more stable. It decoded two bytes every hour and a half approximately. After 13 hours the program cracked the sim cards secret Key Ki. Also the option to attack Ki, was the only function that would work in the program all other function caused a fatal error that crashed the program.



```
SIM_SCAN
Auto
*****
GSM SIM SCAN by Dejan Kaljevic 1999  eMail: dejan@net.yu
URL: http://www.net.yu/~dejan
*****
Ki = 3A 98 37 2B C1 4D 88 E4 EE E3 F4 27 E6 33 8D 92

Success!, Key is tested and it is correct!
```

Figure 42 Ki found by Sim_Scan

It may be possible to speed up this attack 2 or 3 fold by using a faster crystal oscillator and higher baud rate. A sim card will operate up to a clock speed of approximately 11Mhz [9] so a baud rate of 23040 bps and a crystal oscillator running at 8.57MHz. Could be tried or if possible a baud rate of 2880bps and a 10.71 MHz crystal.

1. Conclusion on Comp128 Attack

COMP128 is particularly weak; all the information need to extract Ki form a sim card was readily available on the internet. David Wagner is quoted as saying "There's no way that we would have been able to break the cryptography so quickly if the design had been subjected to public scrutiny, Nobody is that much better than the rest of the cryptography research community [44]." Although the secret key can be compromised in approximately 8 hours you do need physical access to the card.

2. Over-the-air cloning

It has now become clear that to a well equipped attacker an over-the-air attack is possible. GSM experts reported that a number of aspects of the GSM protocols combine to make it possible to mount the mathematical chosen-input attack on COMP128, if one can build a fake base station. Such a fake base station does not need to support the full GSM protocol, and it may be possible to build one with an investment of approximately \$10k [44].

Some technical expertise is probably required to pull off the over-the-air cloning attack, and the attack requires over-the-air access to the target handset for a relatively long period of time. To get around this problem the attack can also be performed in parts: instead of performing an eight-hour attack, the attacker could question the phone for twenty minutes every day on the victim's way to work. Once the SIM is cloned, the SIM-clone is usable until the subscriber gets a new SIM, which in practice does not happen very often. Therefore, *over-the-air cloning must be considered a very real threat*, which should not be ignored. Finally there has been some talk about a new version of COMP128, COMP128/2 but to implement this algorithm every Sim card would have to be replaced. Also Ross Anderson Eli Biham and Lars Knudsen described the merits of Serpent for use on a limited 8-bit processor as is used in smart cards [45]. In the document they describe why Serpent is suitable for the restrictive environment of a smart card where speed is of importance but ram and processing power is very limited. Other stronger one way hash functions could also replace COMP128.

2. Brute Force attack on A5, known plaintext.

Although from the last chapter it is clearly shown that there are much more efficient attacks on A5 than a brute force. I attempted some exhaustive search attacks to see A5's resilience to such an attack. A brute force attack is an attack where each possibility is tried until success is obtained. Typically, a ciphertext is deciphered under different keys until plaintext is recognised. Unfortunately with A5 the plaintext is simply a group of 114 bits so recognising a correct deciphered text is difficult.

3bits	58 bits	26 bits	58 bits	3 bits	8.25 bits
Start	Payload	Training Sequence	Payload	Stop	Guard Period

Figure 43 GSM TDMA structure of the normal burst

Each timeslot within a TDMA frame contains modulated data referred to as a "burst". There are five burst types normal, frequency correction, synchronization, dummy, and access bursts. The normal burst is composed of a 3-bit start sequence, 116 bits of payload, a 26-bit training sequence used to help counter the effects of multi-path interference, a 3-bit stop sequence required by the channel coder, and a guard period. Two bits from the 116-bit payload are used by the Fast Associated Control Channel (FACCH) to signal that a given burst has been borrowed, leaving a total of 114 bits of payload. Contrary to some papers I have read only information in the Payload section is encrypted meaning that we cannot use the training sequences as source of plaintext. During about the first 1/10th of a second of a call the coder will encode silence. A very rough estimate is $13000 \text{ bps} * 0.1 \text{ s} = 1300 \text{ bits}$ of known plaintext [6]. So to attempt to break the conversation your ciphered text will come from the first 24 frames.

Initially I implemented a brute force attack on A5. I took the verified C code for A5 as reverse engineered by Marc Briceno, Ian Goldberg, and David Wagner and generated the first 328 random bits for the key 12 23 45 67 89 AB CD EF and the frame no.134. Then I wrote a loop to generate the output of all keys for a fixed frame no. Starting with the key value 00000000000000. For Each key the program generates the initial 100 bits, which are disregarded, and then the 114 bits for the upstream and also the 114 bits for the downstream. Then the program compares these 228 bits against the previously generated bits for the key, and if they match the session key is compromised.

1. **Generate the first byte then compare**

The second Brute force attack was almost Identical to the first. The only difference came in the comparisons. After generating the 100bits that are disregarded I generated the first byte of the upstream only. I then compared this against the known good output. If they were the different I disregard the key at this stage. There by saving the time need to generate the remaining 220 bits. If the first byte generated was the same as the known good output I would generate the remaining 220 bits of the output and compare then against the previously generate output. This improves the efficiency of the program making it approximately 3 times fast then the initial attack.

2. **Reduce the output**

The next improvement I made was from reading J. Golic, *Cryptanalysis of Alleged A5 Stream Cipher* [12]. The internal state size of A5 is 64 bits. To generate a plaintext attack we only need to know 64 successive bits not the full 228 bits. By reducing the number of bits to generate from 328 to 164 I will reduce the amount of computation necessary. Unfortunately this will not increases the processing speed drastically as only keys that have the same first byte as the session key will benefit from this improvement in the attack. I have reproduced the code for this attack in Appendix E.

3. **C optimisation**

There are numerous methods of optimisation that may be implemented. Techniques such as loop unrolling and strength reduction, it tends to add apparent complexity and hides the core functionality of the program. Function calls for example are a good thing but a function call interrupts an optimiser's train of thought in a drastic way. Any references through pointers or to global variables need to be saved /restored across the function call. Local variable that have had their address taken and passed outside the function also suffer. There is also some overhead to the function call itself as the stack must be manipulated and the program counter altered. I implemented some of these techniques in conjunction with compiling the program in the release and not the Debug version to increases the speed 5 fold. Although the code did bulk up severely.

4. **Finding the best compiler**

Finally I tried several compilers from commercial to freeware to improve the programs speed by a further 25%.

5.

6. **Results and Conclusions**



No.	Description	Key-space	Duration of Search	Keys sec ⁻¹	50% of total keyspace
1	Basic Brute force attack	0x4444000	17H 56m	1110	257312 yrs
2	Generate one byte and compare	0x4444000	6H 27m	3089	92462 yrs
3	Reduce output from 228 to 164 bits	0x4444000	6H 20m	3110	91838 yrs
4	C-Optimisation	0x4444000	1H 38m	11931	23939 yrs
5	Most efficient Compiler	0x4444000	1 H 14m	15950	17907 yrs

Figure 44 Results of a key searches on a single computer

All results listed in this section were attained using a computer with 128Mb of RAM and a 450MHz Pentium II processor running Windows NT. I ran each of the attacks 3 times to remove any deviations. As you can see I reduced the attack almost 260 thousand years to just below 18 thousand years. Although this is almost 14 times faster the attack is still totally unrealistic. Any of the attack describe in the pervious section is much more effective.

```

C:\WINNT\Profiles\9727965\Desktop\641mvstotaloped.exe
70516736 tests so far
70582272 tests so far
70647808 tests so far
70713344 tests so far
70778880 tests so far
70844416 tests so far
70909952 tests so far
70975488 tests so far
71041024 tests so far
71106560 tests so far
71172096 tests so far
71237632 tests so far
71303168 tests so far
71368704 tests so far
71434240 tests so far
71499776 tests so far
71565312 tests so far
key found.
key: 0x1223456789ABCDEF
frame number: 0x000134
known good output(64bits only):
A->B: 0x534EAA58
observed output:
A->B: 0x534EAA58
Elapsed: 4487

```

Figure 45 My fastest attack took 4487 seconds to generate the key space

The attacks were on a key space of 0X 444 0000 (71565312 decimal) keys. I have listed the main steps of the improvement of the attack in the box above. It should also be noted that an increase in processing power will drastically increase the number of keys generated per second but that an increase in RAM does not effect key generation. As

you can see from figure 32 above the processor is working at 100% but less than 6Mbytes of RAM is being used.

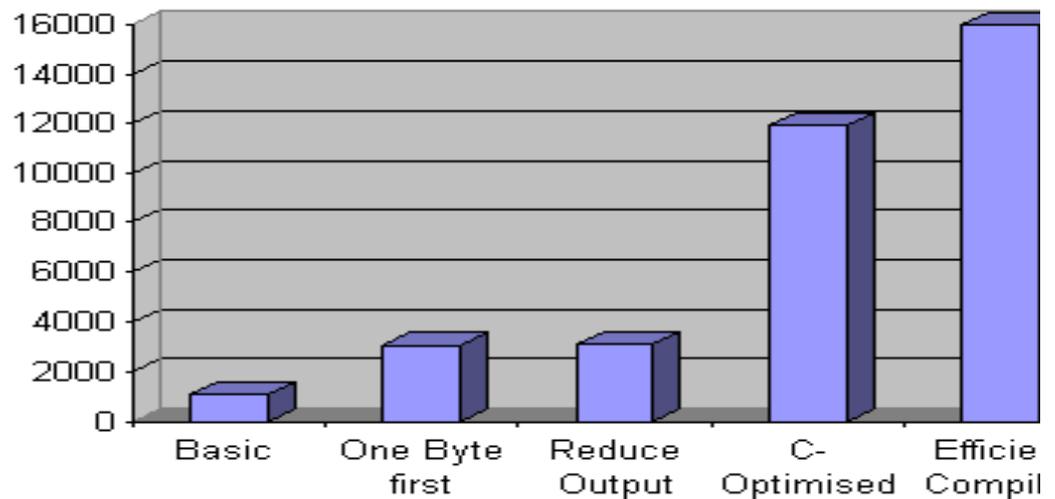


Figure 46 The acceleration of the key search

7. Hardware base attack

I have obtained the verilog code to implement the A5 algorithm (See Appendix D). So it would not be a giant leap to load this onto a Xilinx chip and start testing keys. Hardware attacks are thousands of times faster than software attacks. A chip costing \$10 can be nearly 1000 times quicker than the average home PC running an optimised executable coded in C [1]. A hardware-based attack is the only way to make it feasible to crack A5/1 using a Brute Force attack with currently available computer power. Even porting the attack to a distributed Environment would only marginally decrease the time needed to recover a session key.

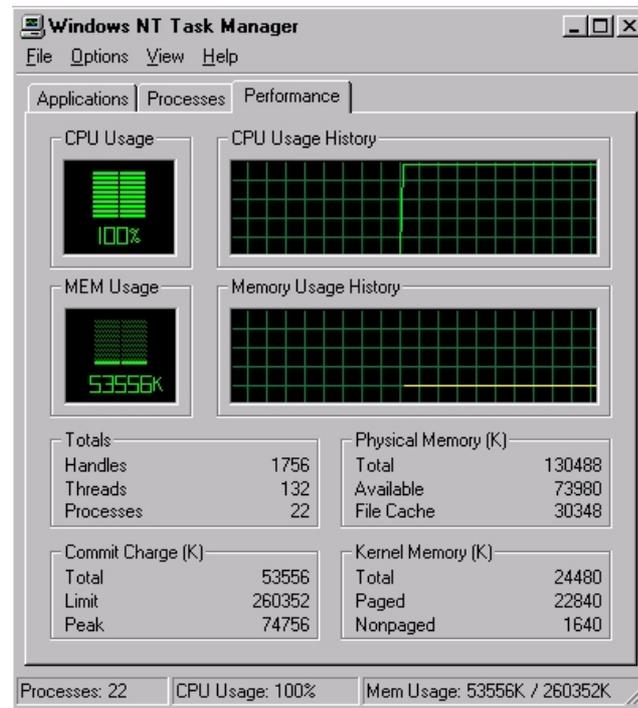


Figure 47 Processing power and Ram being used during an attack

3. Simulation of the GSM Environment

After implementing the attacks above I had all the information necessary to roughly simulate the GSM Environment. This has all the interaction of a really GSM environment but the interaction is through file rather the over the air. Also I am only transmitting the encrypted voice and not a full frame. Initial I took Ki as extracted from the Sim card.

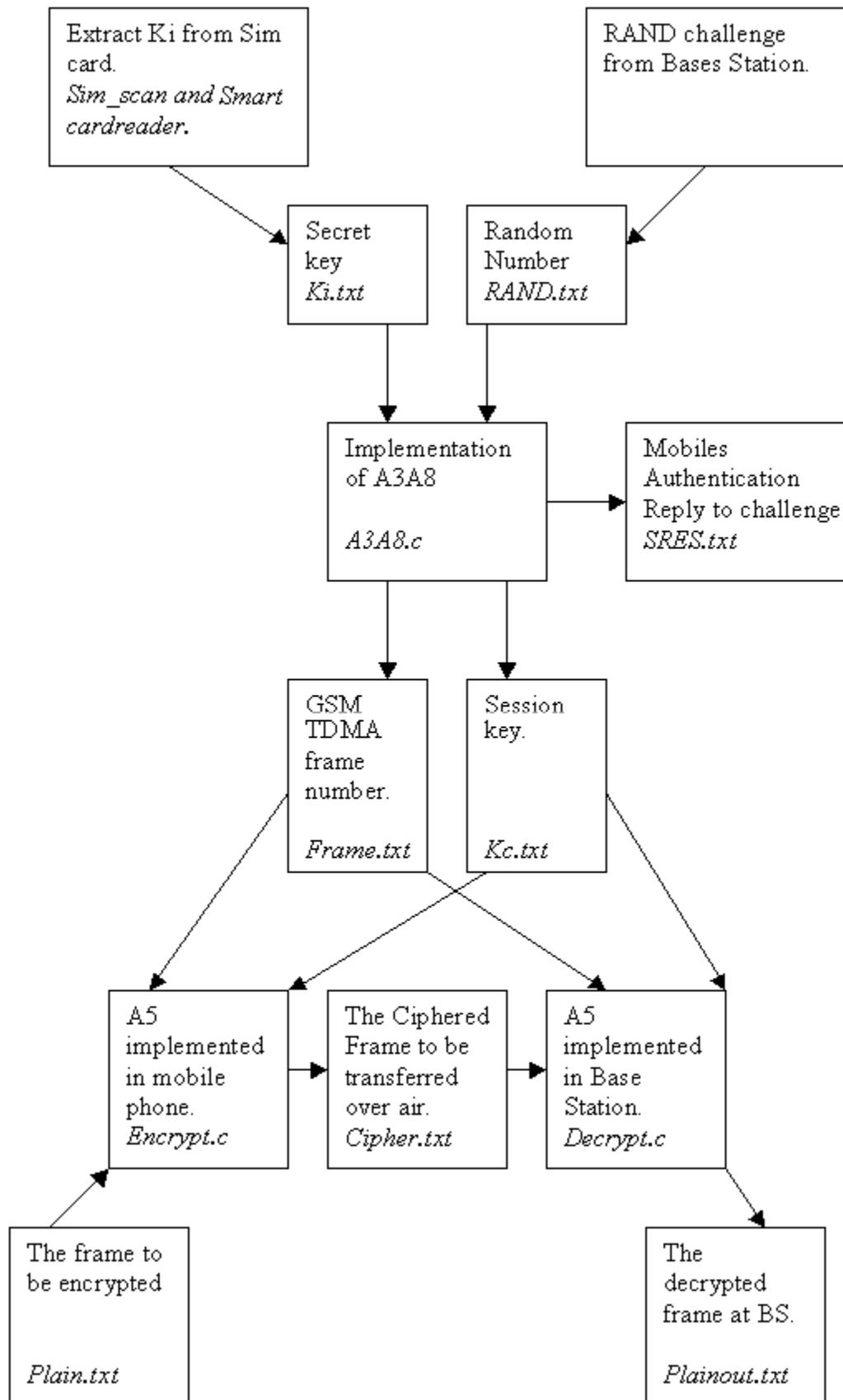


Figure 48 The GSM environment that was implemented

Then I took a random number as would be generated in the Base Station RAND. These two 128-bit numbers are put into text files *Ki.txt* and *RAND.txt* respectively. I use them as input to COMP128 and it produces *Kc.txt* the session key and *SRES.txt* the reply to the Base Stations Random

challenge. Then Kc.txt and the Plaint.txt are ran through A5 to generate a ciphered text that would be sent to the base Station. This ciphered text is used in conjunction with Kc.txt to emulate the decryption of text at the base station to produce Plainout.txt. Although it hardly emulates all the protocols need to implement the GSM service, it does total implement the interaction between the two algorithms use in GSM. Namely the generation of A5's session key by COMP128 from RAND and Ki. But also the synchronisation of the two keystream generators which emulate A5 in both the Mobile Station and the Base Sation.

• Conclusion

In this paper I have shown many flaws in the GSM cryptographic algorithms. I was able to compromise COMP128 and showed numerous flaws in A5 including the fact that it is really only 2^{54} -bit encryption. I believe that these have come from the way that these algorithms where developed in secrecy. In time all were leaked or reverse engineered, versions of A3 A8 A5/1 and A5/2 are all freely available. These algorithms where developed in 1989 – 90 and are just over ten years old. In contrast DES was developed in total openness and released to the entire cryptographic community. It was designed in the mid 70's and remained a government standard for over 20 years. When it was replaced it was replaced by triple DES simply because it's key length was too short but everything else about the algorithm is still strong. The AES has been released in a similar fashion to DES, and is presently being investigated for weaknesses on many fronts.

Essentially the telecom companys need to understand that "security through obscurity" is not the way to develop cryptographic systems. There is also the question of government industrys input into the deliberately weakning of these algorithms. The National Security Agency is known to have pressured the analogous U.S. standards body to weaken voice privacy. Unless consumers demand better, the situation is unlikely to change and we will still only get 54 bit security when we were sold 64 bit, as happened with A5. Both voice-encryption algorithms are flawed, but not obviously so. The attacks on both A5/1 and A5/2 make use of subtle structures of the algorithm, and result in the ability to decrypt voice traffic in real time on average computer equipment. Another point is that if government or police agencies need to acquire legal phone taps they can tap the phone lines at the exchanges where they are not encrypted. The only reason to weaken this system is for "illegal" access. Only wiretaps lacking a court authorization need over-the-air intercepts.

As we look towards 3G you would have thought that the telecom industries would have learnt from their mistakes but as you can see from the following quote from Marc Briceno they have not;

"The GSM MoU still has not recognized the dangers inherent in a secret design process; they have not opened up the design to public review, and in fact they apparently will be designing the next-generation GSM standard (G3) again using this flawed design process. With this decision, we can only expect that further flaws will be discovered in the future. I remain sharply critical of the GSM industry's approach to cryptographic design: it is, in my opinion, irresponsible to consumers and poor scientific practice to boot."

1. **Room for improvement**

The basic ideas behind A5 are good. It is very efficient in the limited space and hardware applications that it is used in. But all of it's weaknesses stem from the it's short registers. Variants of A5 with longer shift registers and denser feedback polynomials should be secure. A5 only uses LFSR where other stream ciphers like RC4 and SEAL are designed more like block ciphers with nonlinear transformations and large S-box etc. Or a block cipher that has been tried and tested could be use in OFB (Output-FeedBack) or CFB (Cipher-FeedBack) mode to get a stream cipher. We have seen how well Serpent Twofish and Rijndael perform in small areas.



Figure 49 Cipher-FeedBack mode

In CFB mode, data can be encrypted in units smaller than the block size. The example above is working on 8-bits. You can encrypt data one bit at a time using 1-bit CFB, which would almost emulate A5, but it seems a lot of encryption for just one bit. Essentially this mode forces a block cipher to be a stream cipher, by using the part of the ciphered text as a function of random number generator. Initially the queue is filled with random data. OFB is similar except that n -bits of the previous block (k_i) are moved into the right-most position of the queue instead of C_i . A5/1 could also be replaced by PIKE, RC4 or SEAL.

• References

1. Bruce Schneier, Applied Cryptography Second Edition, 1996, p 4,189,197-198
2. Chaos Computer Club <http://www.ccc.de>
3. Dejan Kaljevic the author of sim-scans Web page <http://www.net.yu/~dejan/>
4. The GSM System for Mobile communication, Michel Mouly and Marie-Bernadette Pautet, 1992.
5. Alex Biryukov, Adi Shamir, David Wagner, Real Time Cryptanalysis of A5/1 on a PC, April, 2000, presented at the [Fast Software Encryption Workshop 2000](#) New York City
6. M. Briceno, I. Goldberg, D. Wagner, A pedagogical implementation of A5/1, <http://www.scard.org>, May 1999.
7. Mobile Communications, Jochen Schiller, 2000
8. GSM System Security Study,10-1617-01, 10th June 1988 by RACAL RESEARCH LTD.,READING, BERKS. RG2 0SB, ENGLAND. <http://jya.com/gsm061088.htm>
9. HIP Smartcard webpage - <http://cuba.xs4all.nl/~hip/>
10. R. Anderson, M. Roe, A5, <http://jya.com/crack-a5.htm>, 1994.
11. 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.

12. J. Golic, *Cryptanalysis of Alleged A5 Stream Cipher*, proceedings of EUROCRYPT'97, LNCS 1233, pp.239{255, Springer-Verlag 1997.

13. Lauri Pesonen GSM Interception 21.11.1999

<http://www.tml.hut.fi/Opinnot/Tik-110.501/1999/papers/gsminterception/netsec.html>

14. Carl H. Meyer & Stephen M. Matyas, *Cryptography*, 1982 John Wiley & Sons, Inc.

15. <http://www.it.murdoch.edu.au/~hiew/b227/lectures/Topic7-Security.html>

16. <http://www.math.ohiou.edu/~qvu/crypto/9.html>

17. John Savard's *Rijndael* page <http://home.ecn.ab.ca/~jsavard/crypto/co040801.htm>

18. NIST AES press release 9 Aug 1999, Phillip Bullman

20. IBM's submission to the AES Board, September 22 1999.

21. Detailed discription of Mars by crypto <http://home.ecn.ab.ca/~jsavard/crypto/co040806.htm>

22. Mars's home page at IBM <http://www.research.ibm.com/security/mars.html>

23. Twofish's home page <http://www.counterpane.com//twofish.html>

24. Discription RC6 by crypto <http://home.ecn.ab.ca/~jsavard/crypto/co040804.htm>

25. Discription of Serpent by crypto <http://home.ecn.ab.ca/~jsavard/crypto/co040803.htm>

26. AES Fact sheet from the NIST's WebPages. <http://csrc.nist.gov/encryption/aes/aesfact.html>

27. Report on the Development of the Advanced Encryption Standard (AES), By the National Institute of Standards and Technology, October 2, 2000

28. *Dr. Dobb's Journal* December 1998 <http://www.ddj.com/articles/1998/9812/9812b/9812b.htm>

29. Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms by Bryan Weeks, Mark Bean, Tom Rozylowicz, Chris Ficke, National Security Agency

30. What is Differential Cryptanalysis? RSA's FAQ No. 58

<http://www.iks-jena.de/mitarb/lutz/security/cryptfaq/q58.html>

31. What is Linear Cryptanalysis? RSA's FAQ No. 59

<http://www.iks-jena.de/mitarb/lutz/security/cryptfaq/q59.html>

32. M. Matsui and A. Yamagishi. A new method for known plaintext attack of FEAL cipher. In *Advances in Cryptology - Eurocrypt '92*, pages 81-91, Springer-Verlag, 1992.

33. M. Matsui. Linear cryptanalysis method for DES cipher. In *Advances in Cryptology - Eurocrypt*

'93, pages 386-397, Springer-Verlag, 1993.

34. S.K. Langford and M.E. Hellman. Differential-linear cryptanalysis. In *Advances in Cryptology - Crypto '94* , pages 17-25, Springer-Verlag, 1994.

35. B.S. Kaliski Jr. and M.J.B. Robshaw. Linear cryptanalysis using multiple approximations. In *Advances in Cryptology - Crypto '94*, pages 26-39, Springer-Verlag, 1994.

36. RSA FAQ – DES <http://www.rsasecurity.com/rsalabs/faq/3-2-1.html>

37. M.E. Hellman, A cryptanalytic time-memory trade off, *IEEE Transactions on Information Theory* (1980), 401-406.

38. M.J. Wiener, Performance Comparison of Public-Key Cryptosystems, *CryptoBytes* (1) (Summer 1998).

39. E. Biham and A. Shamir, Differential cryptanalysis of the full 16-round DES, *Advances in Cryptology - Crypto '92*, Springer-Verlag (1993), 487-496.

40. RSA FAQ - What are the most important attacks on symmetric block ciphers?

<http://www.rsasecurity.com/rsalabs/faq/2-4-5.html>

41. U. Maurer, Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms, *Advances in Cryptology - Crypto '94*, Springer-Verlag (1994), 271-281.

42. Cracking DES, Secrets of Encryption Research, Wiretap Politics & Chip Design

[By Electronic Frontier Foundation](#), 1st Edition July 1998

43. An implementation of the GSM A3A8 algorithm. (Specifically, COMP128.)

Copyright 1998, Marc Briceno, Ian Goldberg, and David Wagner. All rights reserved.

44. The Berkeley group COMP128 analysis, April 1998 , Ian Goldberg & [Marc Briceno](#)
<http://www.isaac.cs.berkeley.edu/isaac/gsm.html>

45. Serpent and Smartcards, Ross Anderson Eli Biham Lars Knudsen

46. Serge Vaudenay's ``FFT-Hash II is not yet secure"

Further Info,

Shepherd S. J., Cryptanalysis of the GSM A5 Cipher Algorithm, IEE Colloquium on Security and Cryptography Applications to Radio Systems, Digest No. 1994/141, Savoy Place, London, 3 June 1994

Shepherd S. J., An Approach to the Cryptanalysis of Mobile Stream Ciphers, IEE Colloquium on Security and Cryptography Applications to Radio Systems, Digest No. 1994/141, Savoy Place, London, 3 June 1994

I assume both of the documents mentioned above would reveal a lot about the A5 stream cipher and its weaknesses. Unfortunately, Mr Shepherd was convinced not to give either of the above talks and days afterwards both document were classified by the English Government, Mr Shepherd cannot comment on this.

• Glossary

Plaintext: message string to be encrypted.

Ciphertext: encrypted message

Encryption method: algorithm to convert from plaintext to ciphertext.

Encryption Key: a string used in by the encryption method to transform plaintext to ciphertext.

Decryption method: algorithm to convert from ciphertext back to original plaintext.

Decryption Key: a string used in by the decryption method to convert ciphertext back to original plaintext.

Cryptography: the art of coming up with ciphers (methods, keys, etc).

Cryptanalysis: the art of breaking ciphers.

Birthday Paradox: the apparent paradox that, in a schoolroom of only 23 students, there is a 50% probability that at least two will have the same birthday. The "paradox" is resolved by noting that we have a $1/365$ chance of success for each possible pairing of students, and there are 253 possible pairs or combinations of 23 things taken 2 at a time. The overall probability of success is $(1 - 1/365)^{22/2} = 253$ pairs is 0.5005 i.e. 50% chance.

Whitening The very first and last cryptographic operations are some form of mixing of subkeys with the data block in most block ciphers. Whenever this subkey mixing does not naturally occur as the initial step of the first round or the final step of the last round, the subkey mixing as an extra step is called pre- or post-whitening. [27]

VHDL Stands for VHSIC Hardware Description Language. VHSIC is yet another acronym which stands for Very High Speed Integrated Circuits.

• Acknowledgements

I would like to thank the following;

Tom Coffey my supervisor.

Aileen McCarthy for telling me the best ways to corner my supervisor.

Hugh McGuirk for answering all my stupid questions.

Trevor O'Connor, Donal Ward, Noel Guinane, Aongus McGreel, Tomas Winston, James Forde and

Tracey Flannagan.

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.