Cryptanalysis of Akelarre

Niels Ferguson
DigiCash bv
Kruislaan 419
1098 VA Amsterdam, Netherlands
niels@DigiCash.com

Bruce Schneier
Counterpane Systems
101 E Minnehaha Parkway
Minneapolis, MN 55419, USA
schneier@counterpane.com

July 23, 1997

Abstract

We show two practical attacks against the Akelarre block cipher. The best attack retrieves the 128-bit key using less than 100 chosen plaintexts and 2^{42} off-line trial encryptions. Our attacks use a weakness in the round function that preserves the parity of the input, a set of 1-round differential characteristics with probability 1, and the lack of avalanche and one-way properties in the key-schedule. We suggest some ways of fixing these immediate weaknesses, but conclude that the algorithm should be abandoned in favor of better-studied alternatives.

1 Description of Akelarre

Akelarre [AGMP96A, AGMP96B] is a 128-bit block cipher that uses the same overall structure as IDEA [LMM91]; instead of IDEA's 16-bit sub-blocks Akelarre uses 32-bit sub-blocks. Furthermore, Akelarre does not use modular multiplications, but instead uses a combination of a 128-bit key-dependent rotate at the beginning of each round, and repeated key additions and data-dependent rotations in its MA-box (called an "addition-rotation structure" in Akelarre). [1]

Akelarre is defined for a variable-length key and a variable number of rounds. The authors recommend using Akelarre with four rounds and a 128-bit key; this is the version that we will cryptanalyze.

1.1 Encryption

An Akelarre encryption consists of an input transformation, a repeated round function, and an output transformation (see figure 1).

The input transformation is defined as follows:

[1] Data-dependent rotations were first used by Madryga [Mad84] and more recently in RC5 [Riv95].

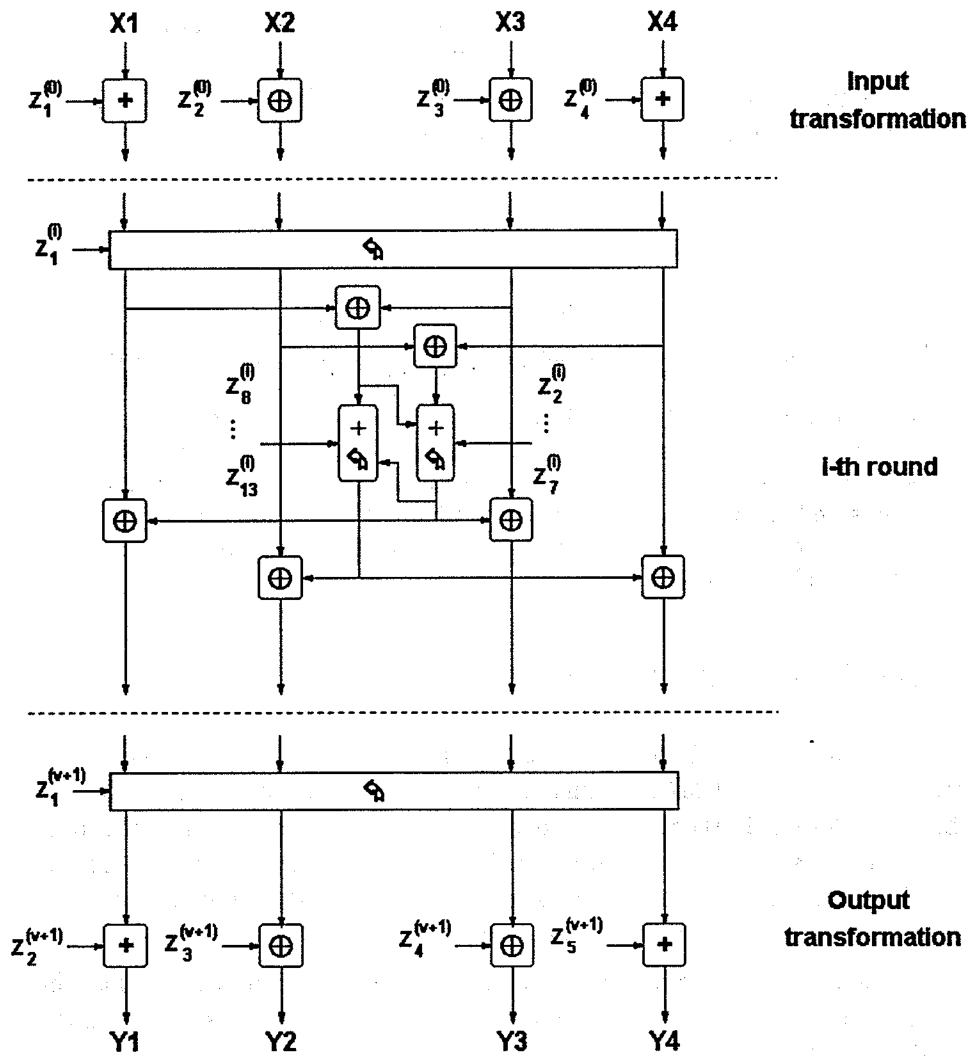

Figure 1: Overview of the Akelarre block cipher

(1) The 128-bit plaintext is divided into four 32-bit sub-blocks: X_1, X_2, X_3, and X_4.

(2) These sub-blocks are combined with four sub-keys (all subkeys are defined as $Z_j^{(i)}$, where i is the round and j indicates the jth sub-key used in round i):

$$\begin{aligned} R_1^{(0)} &:= X_1 + Z_1^{(0)} \bmod 2^{32} \\ R_2^{(0)} &:= X_2 \oplus Z_2^{(0)} \\ R_3^{(0)} &:= X_3 \oplus Z_3^{(0)} \\ R_4^{(0)} &:= X_4 + Z_4^{(0)} \bmod 2^{32} \end{aligned}$$

These four sub-blocks provide the input to round 1.

Akelarre has v rounds. Each round ($i = 1, \ldots, v$) consists of the following steps:

(1) The four input sub-blocks $R_1^{(i-1)}$, $R_2^{(i-1)}$, $R_3^{(i-1)}$, and $R_4^{(i-1)}$ are concatenated into one 128-bit block.

(2) The 128-bit block is rotated left a variable number of bits determined by the least significant seven bits of $Z_1^{(i)}$.

(3) The rotated 128-bit block is divided into four 32-bit sub-blocks: $S_1^{(i)}$, $S_2^{(i)}$, $S_3^{(i)}$, and $S_4^{(i)}$.

(4) Pairs of sub-blocks are XORed to provide inputs to the addition-rotation structure:

$$\begin{aligned} P_1^{(i)} &:= S_1^{(i)} \oplus S_3^{(i)} \\ P_2^{(i)} &:= S_2^{(i)} \oplus S_4^{(i)} \end{aligned}$$

(5) $P_1^{(i)}$ and $P_2^{(i)}$ are combined with twelve 32-bit sub-keys, $Z_2^{(i)}, Z_3^{(i)}, \ldots, Z_{13}^{(i)}$, according to the addition-rotation structure described later. The output of this structure consists of two 32-bit sub-blocks $Q_1^{(i)}$ and $Q_2^{(i)}$.

(6) The four sub-blocks from Step 3 are XORed with the outputs of the addition-rotation structure:

$$\begin{aligned} R_1^{(i)} &:= S_1^{(i)} \oplus Q_2^{(i)} \\ R_2^{(i)} &:= S_2^{(i)} \oplus Q_1^{(i)} \\ R_3^{(i)} &:= S_3^{(i)} \oplus Q_2^{(i)} \\ R_4^{(i)} &:= S_4^{(i)} \oplus Q_1^{(i)} \end{aligned}$$

The sub-blocks $R_1^{(i)}, \ldots, R_4^{(i)}$ form the output of the round function.

The output of the final round forms the input to the output transformation, which consists of the following steps:

(1) The output blocks of the vth round are concatinated into one 128-bit block.

(2) The 128-bit block is rotated left a variable number of bits determined by the least significant seven bits of $Z_1^{(v+1)}$.

(3) The rotated 128-bit block is divided into four sub-blocks: $S_1^{(v+1)}$, $S_2^{(v+1)}$, $S_3^{(v+1)}$, and $S_4^{(v+1)}$.

(4) The four sub-blocks are combined with four final sub-keys:

$$\begin{aligned} Y_1 &:= S_1^{(v+1)} + Z_2^{(v+1)} \bmod 2^{32} \\ Y_2 &:= S_2^{(v+1)} \oplus Z_3^{(v+1)} \\ Y_3 &:= S_3^{(v+1)} \oplus Z_4^{(v+1)} \\ Y_4 &:= S_4^{(v+1)} + Z_5^{(v+1)} \bmod 2^{32} \end{aligned}$$

(5) The four sub-blocks, Y_1, Y_2, Y_3, and Y_4 are concatenated to form the ciphertext.

All that remains is to specify the addition-rotation structure. We describe this for completeness sake; our attack does not rely on any property of the addition-rotation structure. The structure is formed by two columns; $P_1^{(i)}$ is the input to the first column and $P_2^{(i)}$ is the input to the second column. Each column works as follows:

(1) The high 31 bits of $P_j^{(i)}$ are rotated left a variable number of bits.

(2) The 32-bit output of the previous step is added to a sub-key.

(3) The low 31 bits of the result of the previous step are rotated left a variable number of bits.

(4) The 32-bit output of the previous step is added to a sub-key.

(5) The high 31 bits of the result of the previous step are rotated left a variable number of bits.

(6) The 32-bit output of the previous step is added to a sub-key.

(7) Steps 3 through 6 are repeated until there have been seven rotations and six sub-key additions total.

(8) The outputs of the two column are $Q_1^{(i)}$ and $Q_2^{(i)}$.

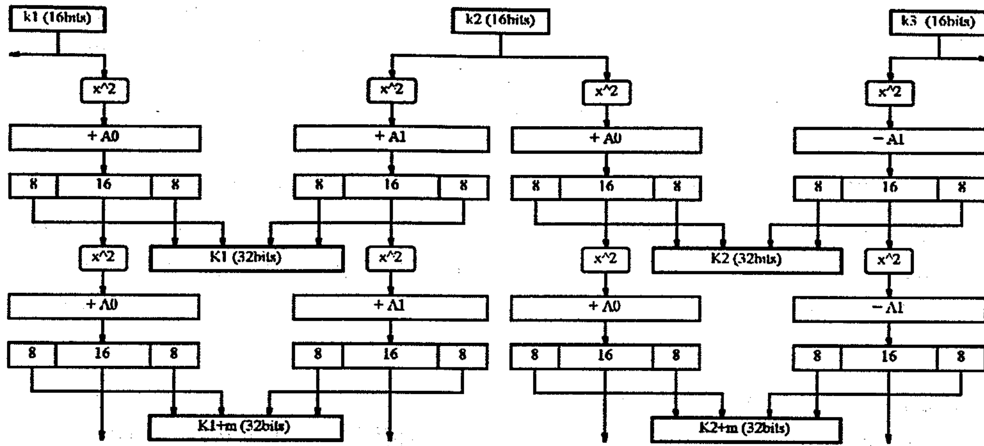

Figure 2: Overview of the Akelarre key schedule

The sub-keys added in the first column are $Z_8^{(i)}, Z_9^{(i)}, \ldots Z_{13}^{(i)}$; the sub-keys added in the second column are $Z_2^{(i)}, Z_3^{(i)}, \ldots, Z_7^{(i)}$.

Let $X[a..b]$ be the number formed by taking bits a through b from the integer X (where we start our bit numbering at 0 for the least significant bit). The rotation amounts of the second column are determined by $P_1^{(i)}$: the first rotation amount is $P_1^{(i)}[4..0]$, the second rotation amount is $P_1^{(i)}[9..5]$, the third rotation amount is $P_1^{(i)}[14..10]$, the fourth rotation amount is $P_1^{(i)}[19..15]$, the fifth rotation amount is $P_1^{(i)}[23..20]$, the sixth rotation amount is $P_1^{(i)}[27..24]$, and the seventh rotation amount is $P_1^{(i)}[31..28]$. The rotation amounts in the first column are determined in the same manner from $Q_2^{(i)}$.

1.2 Key Schedule

Akelarre requires $13v+9$ sub-keys (four for the input transformation, 13 for each of the v rounds, and five for the output transformation). These 32-bit sub-keys are derived from a master key. The length of the master key can be any multiple of 64 bits, although we limit our discussion to 128-bit master keys, which is the key size suggested in [AGMP96A]. The description of the key schedule in [AGMP96A] and [AGMP96B] are different; we base our discussion on the more extensive description in [AGMP96A].

An overview of the key schedule is shown in figure 2. First, the master key is divided into eight 16-bit sub-blocks, called k_i for $i = 1, \ldots, 8$. Each sub-block is squared (yielding a 32-bit result), and then added mod 2^{32} to a constant, $A_0 = \text{A49ED284}_{(16)}$ and $A_1 = \text{735203DE}_{(16)}$. Let $k_i^{(1)} := k_i^2 + A_0 \bmod 2^{32}$ and $k_i^{(1')} := k_i^2 + A_1 \bmod 2^{32}$.

The first eight sub-keys are generated as follows: The outermost bytes of $k_i^{(1)}$

form the two high-order bytes of sub-key K_i; the outermost bytes of $k^{(1')}_{(i \bmod 8)+1}$ form the two low-order bytes of sub-key K_i. Thus, sub-key K_i is a function of only k_i and $k_{(i \bmod 8)+1}$.

The innermost bytes of $k^{(1)}_i$ are squared and added modulo 2^{32} to A_0 to generate $k^{(2)}_i$, and similarly the innermost bytes of $k^{(1')}_i$ are squared and added modulo 2^{32} to A_0 to generate $k^{(2')}_i$. The second eight sub-keys are generated in the same way the first eight were. For $i = 9, \ldots, 16$, the outermost bytes of $k^{(2)}_{i-8}$ form the two high-order bytes of sub-key K_i; the outermost bytes of $k^{(2')}_{(i \bmod 8)+1}$ form the two low-order bytes of sub-key K_i.

This process is repeated, every round of the key schedule squares the middle bytes of the $k^{(j)}_i$ and $k^{(j')}_i$ values and generates 8 additional sub-keys, untill all 61 required sub-keys have been generated.

After calculating all the K_i sub-keys, they are read sequentially to fill the $Z^{(i)}_j$ keys required for encryption; decryption keys are derived from these keys as required.

2 Cryptanalysis of Akelarre

The pivotal observation is that the round function preserves the parity of the input. The 128-bit rotate does not influence the parity. The subsequent addition-rotation structure XORs each of its outputs twice into the data blocks, thus preserving parity. The only operations in Akelarre that affect the parity of the input are the input transformation and the output transformation. This allows us to attack the key blocks involved in those transformations irrespective of the other properties of the round function.

We implement a chosen plaintext attack in four phases. In the first phase, we find most of the bits of two of the sub-keys of the output transformation. In the second phase, we find most of the bits of two of the sub-keys of the input transformation. In the third phase, we exploit the key schedule to recover 80 bits of information about the master key. In the fourth phase, we exhaustively search through all remaining possible master keys.

2.1 Recovering Output Transformation Sub-Key Bits

We start by fixing $X_1 = 0$ and $X_4 = 0$, and encrypting many blocks with random values for X_2 and X_3. Let $\mathcal{P}(\cdot, \cdot, \ldots)$ denote the parity of the concatenation of all its arguments (sum all the bits modulo 2). We define:

$$
\begin{aligned}
k &:= \mathcal{P}(Z^{(0)}_1, Z^{(0)}_2, Z^{(0)}_3, Z^{(0)}_4) \\
x &:= \mathcal{P}(X_2, X_3) \\
r &:= \mathcal{P}(R^{(0)}_1, \ldots, R^{(0)}_4)
\end{aligned}
$$

It is easy to see that $r = k \oplus x$.

As the round function is parity-invariant, we have $r = \mathcal{P}(R_1^{(v)}, \ldots, R_4^{(v)})$ after v rounds, and thus $r = \mathcal{P}(S_1^{(v+1)}, \ldots, S_4^{(v+1)})$.

Let $K_1 := -Z_2^{(v+1)} \bmod 2^{32}$, and $K_4 := -Z_5^{(v+1)} \bmod 2^{32}$. This gives us

$$r = \mathcal{P}((Y_1 + K_1) \bmod 2^{32}, Y_2 \oplus Z_3^{(v+1)}, Y_3 \oplus Z_4^{(v+1)}, (Y_4 + K_4) \bmod 2^{32})$$

Collecting all our formulae, we get

$$\mathcal{P}((Y_1 + K_1) \bmod 2^{32}, (Y_4 + K_4) \bmod 2^{32}) = k' \oplus x \oplus y \qquad (1)$$

where $k' := k \oplus \mathcal{P}(Z_3^{(v+1)}, Z_4^{(v+1)})$ and $y := \mathcal{P}(Y_2, Y_3)$. We define for any K, $K^* := K[30..0]$ to be the number formed by the least significant 31 bits of K. By splitting of the most significant bits of the sum we can rewrite equation 1 as

$$\mathcal{P}(Y_1^* + K_1^*, Y_4^* + K_4^*) = k'' \oplus x \oplus y' \qquad (2)$$

where $k'' := k' \oplus K_1[31] \oplus K_4[31]$ and $y' := y \oplus Y_1[31] \oplus Y_4[31]$. The value k'' depends only on the key, and will be the same for all of our encryptions. The values x and y' are known, as they only depend on the plaintext or ciphertext.

If we find two encryptions i and j which have the same value for Y_1^* (i.e. $Y_{1,i}^* = Y_{1,j}^*$), then we can derive a sum-parity relation for K_4^*. We get

$$\mathcal{P}(Y_{4,i}^* + K_4^*) \oplus \mathcal{P}(Y_{4,j}^* + K_4^*) = x_i \oplus x_j \oplus y_i' \oplus y_j' \qquad (3)$$

Such an equation eliminates about half of the possible values for K_4^*. After $4 \cdot 10^5$ chosen plaintexts, we can expect about 37 separate collisions for Y_1^*, and thus about 37 sum-parity relations for K_4^*. We can now exhaustively search the 2^{31} possible values of K_4^* for a value that satisfies all of the parity relations. Numerical experiments indicate that 37 relations are usually enough to give a unique solution. Once K_4^* has been found, every encryption that was done provides an equivalent sum-parity relations for K_1^*, which allows us to exhaustively search for K_1^*. (The order can of course be reversed, with collisions on Y_4^* giving sum-parity relations for K_1^*, which allows us to recover K_1^* first.)

Overall, this phase of the attack requires about $4 \cdot 10^5$ chosen plaintexts, and 2^{32} exhaustive search steps to recover both K_1^* and K_4^*. Several refinements are possible. The key schedule cannot generate all 2^{32} possible sub-keys; this information can be used to speed up the exhaustive search. As will be obvious from the key schedule, the possible sub-key values can be enumerated by listing the possible values for the two halves of the sub-key separately. This results in about 2^{25} possible values for the least significant 31 bits of the sub-keys in the output transformation. (This assumes a 4-round Akelarre. Due to the nature of the key schedule, the entropy of the sub-keys in the output transformation decreases as the number of rounds increases.)

The last phase in our attack is an exhaustive search over 2^{48} possible master keys (see section 2.4), which requires a complete Akelarre encryption per possible master key. Checking 2^{50} possible key values using sum-parity relations is certainly going to be a lot less work. This leads to the following improvement: Using only 60 chosen plaintexts, we search for for K_1^* and K_4^* in parallel using equation 2. There are about 2^{25} possible values for each of these two values, which gives us a total of 2^{50} possible values for the pair. We can expect to find the right values (that satisfy all the sum-parity relations) in about 2^{49} tries. The computational effort in this phase is still negligable compared to the effort required in the last phase of our attack, as each of the operations in this phase is far less complex.

The search can be improved even further if we take the non-uniformity of the key-block distribution into account. From the key schedule it is easy to derive the probabilities for each of the 2^{25} possible sub-keys. This can be done by computing independent probabilities for each of the two halves of the sub-keys. Our results indicate that this leaves about 23.5 bits of entropy for each of the K^* values. By searching the high-probability values first we can expect to find the correct key values sooner.

2.2 Recovering Input Transformation Sub-Key Bits

We can recover the 31 least significant bits of $Z_1^{(0)}$ and $Z_4^{(0)}$ as well. We could, of course, perform the analysis from the previous section on the decryption function, but there are much more direct methods.

Once we have recovered K_1^* and K_4^*, we can recognise whether two encryptions have the same parity during the rounds. (We can decrypt enough of the output transform; the key bits that we don't know affect the parity in the same way for each encryption.) Choose fixed values for X_1, X_2, and X_3, and perform encryptions for different values of X_4. This gives us sum-parity relations for $Z_4^{(0)*}$ similar to equation 3. Using the same methods as in the previous step, we can thus recover the 31 least significant bits of $Z_4^{(0)}$, and $Z_1^{(0)}$, using 2^{32} exhaustive search steps and about 80 chosen-plaintexts.

A more direct method is also possible, where every chosen plaintext encryption reveals one bit of $Z_4^{(0)*}$ or $Z_1^{(0)*}$. This eliminates the exhaustive searches for these 31-bit values, and reduces the number of chosen-plaintexts for this phase to 62. The details of this method are left as an excersise to the reader.

2.3 Recovering Master Key Information from the Sub-Keys

We have recovered the 31 least significant bits of 4 of the sub-keys. Due to the structure of the key schedule, each half of a sub-key depends on exactly 16 bits of the master key.

Table 1 give the expected information provided by the partially known sub-keys about the master key blocks, assuming that the master key is chosen uniformly

Sub-key	upper half	lower half
$Z_1^{(0)*}$	11.99 bits about k_1	12.85 bits about k_2
$Z_4^{(0)*}$	11.99 bits about k_4	12.85 bits about k_5
$Z_2^{(5)*}$	11.52 bits about k_2	12.01 bits about k_3
$Z_5^{(5)*}$	11.52 bits about k_5	12.01 bits about k_6

Table 1: Bits of information provided by sub-key about master sub-keys

at random. As the mapping from a master key block to one half of a sub-key is not bijective, not all 2^{16} possible values of the sub-key half can occur. In fact, each 32-bit sub-key has between 24.1 and 25.7 bits of entropy.

Some of the master key blocks influence two of the recovered sub-keys. In this case we can expect to be left with a single possible value for this master key block. (As there are only 16 bits in a master key block, we can't have more than 16 bits of information about it.)

An interesting observation is that the amount of information that we get about the master key depends eratically on the number of rounds, due to the alignment of the known sub-keys in the key schedule. In some cases the known sub-keys are all derived from 4 of the master key blocks, while in other cases they are derived from 7 master key blocks. If we increase the number of rounds to 5, we can expect to get about 7 bits more information about the master key blocks, making the 5-round Akelarre significantly weaker against our attack than the 4-round version.

2.4 Recovering the Entire Master Key

Adding up the information that we get, we can expect to have 80 bits of information about the 128-bit key. This leaves about 2^{48} possible master key values. These are easy to enumerate: For each master key block we create a list of all possible values. For those master key blocks that influence some of the known sub-keys, we try all 2^{16} possible values and discard those that don't match the known sub-key bits. We will be left with 2 master key blocks that are fully known, 4 master key blocks that are partially known, and 2 master key blocks that are unknown. The cartesian product of these 8 lists enumerates the possible values for the master key.

Using an exhaustive search over these possible master key values, we can expect to find the entire 128-bit master key after at most 2^{48} tries, with an expected workload of 2^{47} tries.

3 A second attack

Our second attack uses the observation that the Akelarre round function has a lot of excellent differential characteristics. In fact, any 64-bit pattern repeated once to form a 128-bit word gives a differential 1-round characteristic with probability 1, and the output differential is a rotation of the input differential. Thus, the Akelarre round function has 2^{64} 1-round differential characteristics with probability 1.

The set of differences we are particularly interested in are those with exactly 2 one bits, where the bits are 64 bit-positions apart. If such a differential occurs during the rounds we can easilly detect this from the ciphertext. So if we use an input differential that flips one bit in X_3 and the corresponding bit in X_1, we can detect if the flipped bit in X_1 resulted in the same bit being flipped in the output of the input transformation. This gives us one bit of information about the first key block of the input transformation.

Using 63 chosen plaintexts, we can recover the same 62 bits of information about the key of the input transformation as we did in the previous attack, but now without any exhaustive searching. Once we have these key bits, we can generate all 62 differentials we are interested in, and use these to recover the 62 bits of the output transformation key we found in the first attack, again without exhaustive searching. Furthermore, we can observe the sum effect of all the 128-bit rotates modulo 64, which gives us 6 more bits of information about the expanded key. Using some fairly straightforward precomputations this reduces the work load of the exhaustive master-key search by a factor of 64, giving us a maximum of 2^{42} tries and 2^{41} tries on average before the key is found.

As about half of our differential attempts in the first half of this attack resulted in the desired differential pattern during the rounds, we don't have to regenerate all 62 interesting differentials to find the 62 key bits of the output transformation, but (on average) only 31 of them. This reduces the expected number of required plaintexts to less than 100.

Further refinements are possible if we use the fact that the output transformation key blocks are not independent of the input transformation key blocks. Using this information, we can further reduce the number of required plaintexts.

4 Fixing Akelarre

There are three obvious weaknesses in Akelarre that we exploited in our attack. The round function is parity-preserving, which allows us to attack the input and output transformation keys irrespective of the complexity of the addition-rotation structure, and irrespective of the number of rounds. The only elementary operation that Akelarre employs that is not parity-preserving is the addition modulo 2^{32}. Replacing the XORs used to mix the output of the addition-rotation structure with the data blocks by additions would eliminate this property.

The differential characteristics again work irrespective of the number of rounds or the complexity of the addition-rotation structure. These differential characteristics can be broken up by replacing the rotation at the beginning of a round with a different function that does not preserve our characteristic patterns.

The key schedule is especially weak. Learning one bit of any sub-key gives immediate information about the master-key, although the designers state that the key schedule was explicitly designed to avoid this property. The main problem is the use of 16-bit blocks without any diffusion between the key blocks. The 16-bit block size does not allow any one-wayness properties. The only fix would seem to design an entirely new key schedule. One possible solution is to derive the sub-keys from a cryptographically strong pseudo-random generator which uses the master key as seed.

Even with these fixes it is unclear how strong the fixed Akelarre cipher would be.

5 Conclusions

For a 128-bit block cipher, Akelarre is disappointingly weak. The amount of work necessary for a successful attack is three or four orders of magnitude less than that of attacking DES. As such, Akelarre is not suitable for applications that require even a medium level of security. And while the algorithm may be repairable, it does not offer any obvious speed advantages over more established alternatives.

The weaknesses that we have found do not inspire confidence in the design process used to create Akelarre. Even if all these weaknesses were to be fixed, the resulting cipher would still be tainted by an apperently ad-hoc design process and leave doubt about other as yet undiscovered weaknesses. Therefore, we recommend that the Akelarre design be abandoned.

Since the original publication the authors have published a new version with an improved key schedule [AGMP97]. We have not investigated this new version in any depth, but even the improved key schedule allows us to recover 31 bits of information about the master key in a trivial manner.

6 Acknowledgements

We would like to thank Fausto Montoya for providing us with the figures describing Akelarre.

References

[AGMP96A] G. Álvarez, D. de la Guía, F. Montoya, and A. Peinado, "Akelarre: a new Block Cipher Algorithm," *Third Annual Workshop on Selected*

Areas in Cryptography (SAC '96), Kingston, Ontario, 15–16 August 1996, pp. 1–14.

[AGMP96B] G. Alvares Marañón, D. de la Guía Martínez, F. Montoya Vitini, and Alberto Peinado Domínguez, "Akelarre: Nuevo Algoritmo de Cifrado en Bloque," *Actas de la IV Reunión Española Sobra Criptología*, Universidad de Valladolid, September 1996, pp. 93–100. (In Spanish.)

[AGMP97] G. Álvarez, D. de la Guía, F. Montoya, and A. Peinado, "Description of the new Block Cipher Algorithm Akelarre", `http://www.iec.csic.es/~fausto/papers/akelarre1.ps`

[LMM91] X. Lai, J. Massey, and S. Murphy, "Markov Ciphers and Differential Cryptanalysis," *Advances in Cryptology—CRYPTO '91*, Springer-Verlag, 1991, pp. 17–38.

[Mad84] W.E. Madryga, "A High Performance Encryption Algorithm," *Computer Security: A Global Challenge*, Elsevier Science Publishers, 1984, pp. 557–570.

[Riv95] R.L. Rivest, "The RC5 Encryption Algorithm," *Fast Software Encryption, Second International Workshop Proceedings*, Springer-Verlag, 1995, pp. 86–96.

Two Rights Sometimes Make a Wrong

Lars R. Knudsen Vincent Rijmen*

Katholieke Universiteit Leuven, ESAT-COSIC

K. Mercierlaan 94, B-3001 Heverlee, Belgium

lars.knudsen@esat.kuleuven.ac.be

vincent.rijmen@esat.kuleuven.ac.be

Abstract

At the SAC'96 a new iterated block cipher, Akelarre, was proposed. Akelarre uses components of the block ciphers RC5 and IDEA and is conjectured strong with four rounds. This paper shows that Akelarre with any number of rounds is weak even under a ciphertext only attack. This illustrates that mixing two (presumably) strong ciphers is not always a good idea.

1 Introduction

At the SAC'96 a new block cipher, Akelarre, was proposed [1]. Akelarre is an iterated cipher, which uses components of the block ciphers RC5[4] and IDEA[2]. A comparison is made to these block ciphers in favor of Akelarre. In the following we will show that Akelarre is a weak block cipher with any number of rounds. The paper is organised as follows. § 2 contains a

*F.W.O. research assistant, sponsored by the Fund for Scientific Research - Flanders (Belgium)

short description of Akelarre with the details necessary for our attacks; we refer to [1] for the full description. In § 3 we describe the main weakness of Akelarre, which forms the basis of our attacks. In § 4 a known plaintext attack and a ciphertext only attack are given and § 5 contains our concluding remarks.

2 Description of Akelarre

Akelarre [1] is a 128-bit block cipher. The key length is variable, but always a multiple of 64 bits. Akelarre has a structure similar to IDEA [2]. The main differences are the following.

- Akelarre uses 32-bit words instead of 16-bit words.
- The multiplication-addition structure is replaced by a complex addition-rotation structure (AR-structure).
- No modular multiplications are used.
- In the round transformation of IDEA there are key additions inside and outside the MA-structure, in Akelarre there are only key additions in the AR-structure.
- The key scheduling is more complicated and difficult to invert.

The AR-structure of Akelarre consists of 12 31-bit data dependent rotations and 12 key additions. This structure is reminiscent of the RC5 [4] round operation. Akelarre has an input transformation, an output transformation and a variable number of rounds. It is proposed with four rounds. Figure 1 shows Akelarre with one round. To simplify notation a different numbering for the round keys has been adopted. Our attacks are independent of the 12 round keys in each AR-structure, which therefore are denoted Z_r for round r.

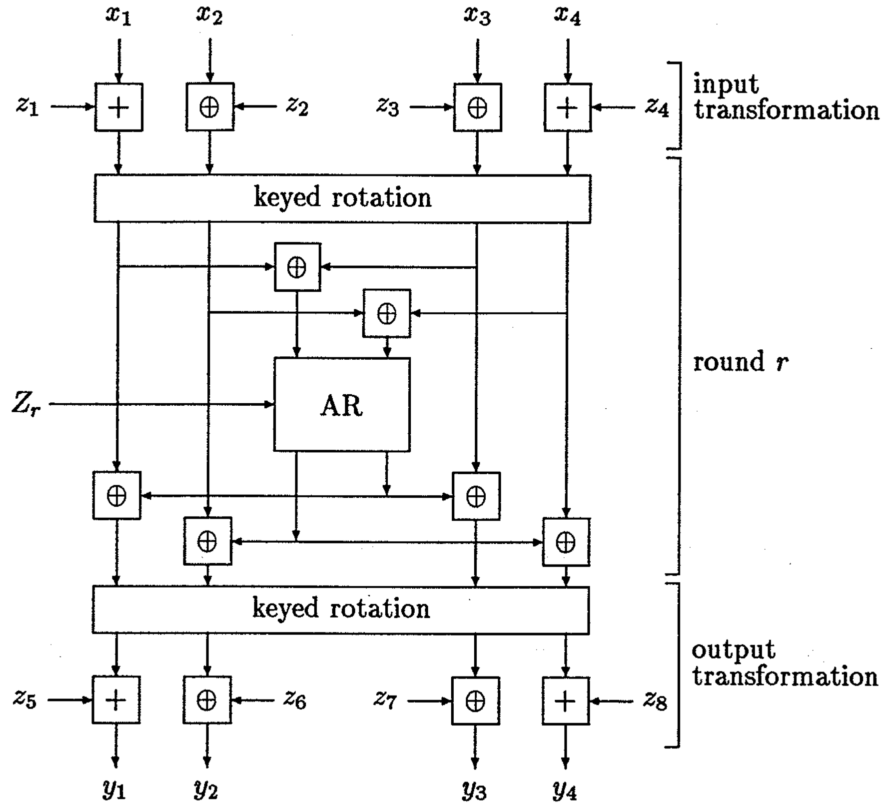

Figure 1: Computational graph of Akelarre.

The plaintext is denoted $x_1 \| x_2 \| x_3 \| x_4$. where '$\|$' denotes concatenation of bit strings. The input transformation consists of adding modulo 2^{32} respectively exoring the key words $z_1, \ldots, z_4$ to the plaintext, according to Figure 1. The input of round r will be denoted $x_1^r \| x_2^r \| x_3^r \| x_4^r, r = 1, \ldots R$, The round transformation for round i is as follows.

$$u_1^i \| u_2^i \| u_3^i \| u_4^i = \mathrm{rot}_{r^i}(x_1^i \| x_2^i \| x_3^i \| x_4^i) \tag{1}$$

$$(t_1, t_2) = \mathrm{AR}(u_1^i \oplus u_3^i, u_2^i \oplus u_4^i) \tag{2}$$

$$x_1^{i+1} = u_1^i \oplus t_1 \tag{3a}$$

$$x_2^{i+1} = u_2^i \oplus t_2 \quad (3b)$$

$$x_3^{i+1} = u_3^i \oplus t_1 \quad (3c)$$

$$x_4^{i+1} = u_4^i \oplus t_2 \quad (3d)$$

where rot_{r^i} is a key dependent 128-bit rotation by r^i positions. The inputs of the output transformation are denoted x_i^{R+1}. The output transformation consists of adding modulo 2^{32} respectively exoring the key words $z_5, \ldots, z_8$.

3 Weakness of the Round Transformation

In this section we describe the main observation to be used in our attacks. The round transformation of Akelarre exhibits an invariant relation between the input and the output.

$$\begin{aligned}(x_1^{i+1} \oplus x_3^{i+1}) \,\|\, (x_2^{i+1} \oplus x_4^{i+1}) &= (u_1^{i+1} \oplus u_3^{i+1}) \,\|\, (u_2^{i+1} \oplus u_4^{i+1}) \\ &= \text{rot}_{r^i \bmod 64}((x_1^i \oplus x_3^i) \,\|\, (x_2^i \oplus x_4^i))\end{aligned}$$

After R rounds this is

$$(x_1^{R+1} \oplus x_3^{R+1}) \,\|\, (x_2^{R+1} \oplus x_4^{R+1}) = \text{rot}_{s^{R-1}}((x_1^1 \oplus x_3^1) \,\|\, (x_2^1 \oplus x_4^1)),$$

where $s^R = (\sum_{i=1}^{R} r^i) \bmod 64$. The input and output transformation destroy this invariant. Let $s^\rho = s^R + r^{R+1}$, where the last term denotes the rotation amount in the output transformation. For the whole block cipher the relation is

$$((y_1 - z_5) \oplus y_3 \oplus z_7) \,\|\, ((y_4 - z_8) \oplus y_2 \oplus z_6)$$

$$= \mathrm{rot}_{s^\rho}(((x_1 + z_1) \oplus x_3 \oplus z_3) \,||\, ((x_4 + z_4) \oplus x_2 \oplus z_2)), \quad (4)$$

If the keys $z_i, i = 1, \ldots 8$ and s^ρ are known, it is possible to calculate the exor of two halves of any plaintext block from the exor of the two halves of the corresponding ciphertext block. This situation is comparable to an encryption with a one time pad where the key is used twice. If the plaintext contains enough redundancy, it can be uniquely determined.

As we will show in the following section, it is possible to determine the keys of (4). Once these keys have been determined the attacker gets immediate information about the plaintexts from intercepted ciphertexts. We first describe a known plaintext attack then a ciphertext only attack. Both attacks assume that some statistics of the plaintext are known. If the encrypted text is an English text, a LaTeX document, or even consists of random ASCII-characters this gives enough redundancy to recover the key.

4 Cryptanalysis of Akelarre

We describe two attacks that do not recover the complete Akelarre key, but give enough information about the key to allow the cryptanalyst to recover the plaintexts from the ciphertexts.

4.1 A Known Plaintext Attack

4.1.1 Recovering the keys

Equation (4) is an equation in 64 bits, containing one unknown rotation and eight unknown 32-bit key words. Five known plaintexts and their ciphertexts give enough information to solve the equations for the unknown key bits. This can be done very efficiently by guessing a value for s^R and then solve for the z_i's, starting from the least significant bits. Four known

plaintexts are used to solve the equations and the fifth to verify. Since s^R can take only 128 different values, the work factor of this approach is very small.

The keys z_1, z_4, z_5, z_8 and s^R can be determined uniquely. Since only exor information is available, it is not possible to determine z_2, z_3, z_6 and z_7 uniquely. Depending on the value of s^R it is possible to determine $z_2 \oplus z_6$ and $z_3 \oplus z_7$, or $z_2 \oplus z_7$ and $z_3 \oplus z_6$.

4.1.2 Recovering plaintexts

After recovering the keys as described in the previous section, the cryptanalyst can examine new ciphertexts and try to recover the plaintexts from them. This will only be possible if the plaintext contains some redundancy. To simplify the discussion we assume that $s^R = 0$. Equation (4) is then:

$$(x_1 + z_1) \oplus x_3 = (y_1 - z_5) \oplus y_3 \oplus z_7 \oplus z_3 \tag{5a}$$

$$(x_4 + z_4) \oplus x_2 = (y_4 - z_8) \oplus y_2 \oplus z_6 \oplus z_2 . \tag{5b}$$

The right hand sides of (5) are known. A cryptanalyst who tries to determine $x_1 \| x_2 \| x_3 \| x_4$, faces a problem that has a strong resemblance to the decryption of a one time pad where the key has been used twice. The situation is a bit more complex, because there are actually two pads used, one for the even numbered words and one for the odd numbered words. If the right hand sides are denoted k_1 and k_4, the plaintexts are given by

$$x_1 \| x_2 \| x_3 \| x_4 = a \| b \| ((a + z_1) \oplus k_1) \| ((b \oplus k_4) - z_4) ,$$

where a and b can take every value. If the plaintext contains enough redundancy, this problem can be solved. Even if the redundancy of the plaintext is small, there is a leak of plaintext information to the ciphertext.

4.2 A Ciphertext Only Attack

It is possible to recover the eight key words z_i and s^R using only statistical information on the distribution of the plaintext. Afterwards the approach of the previous section can be used to recover plaintexts.

4.2.1 Recovering s^R

If the plaintext consists of ASCII characters, the most significant bit of every byte will be zero (or with probability close to one for an extended set of ASCII characters). In the following we assume that the most significant bit of every byte is a zero bit. Exoring of some bytes with the bytes of the keys z_2, z_3, z_6 and z_7 will keep these most significant bits constant. Even after the addition of z_1, z_4, z_5 and z_8 these bits will be biased with a high probability. By observing the ciphertexts it is possible to see where the almost constant bits have moved to. In this way s^R mod 8 can be determined. Computer experiments have shown that with 5000 ciphertexts the success rate is close to 1.

Once s^R is known modulo eight, there are four possibilities left modulo 32. The rest of the attack is simply repeated for every possible value. Note that the addition becomes less and less linear for more significant bits. This probably allows to determine s^R modulo 32 in a more efficient way than just guessing.

4.2.2 Recovering the z_i keys

Once the rotation modulo 32 is known, it can be partially compensated for by applying the inverse rotation to the ciphertexts and the keys of the output transformation. In the further analysis we assume that $s^R = 0 \bmod 32$ because this makes the discussion much easier to follow. Equation (4) can be written as follows.

$$(y_1 - z_5) \oplus y_3 \oplus z_7 = (x_v + z_v) \oplus x_{v_2} \oplus z_{v_2} \tag{6a}$$

$$(y_4 - z_8) \oplus y_2 \oplus z_6 = (x_w + z_w) \oplus x_{w_2} \oplus z_{w_2} \tag{6b}$$

The parameters (v, v_2, w, w_2) can take the values $(1, 3, 4, 2)$ and $(4, 2, 1, 3)$. The exact value depends on s^R. We assume that $(v, v_2, w, w_2) = (1, 3, 4, 2)$. Since the attack uses only information on the distribution of the plaintexts and not on their actual value, it will also work if this assumption is wrong. The only visible effect will be that the keys have been labeled erroneously. When the keys are used to recover plaintext both possibilities should be checked.

The attack takes one equation of (6) at a time. We assume here that the plaintext consists of bytes with the most significant bit equal to 0. In this case it is possible to recover the keys byte by byte, starting with the least significant byte. In the remainder of the analysis the variables y_i, z_i, x_i will stand for the byte that is examined.

The cryptanalyst uses the information he has on the distribution of the plaintext bytes to build an off-line table that contains for every value of z_1 the distribution of $(x_1 + z_1) \oplus x_3$. These distributions are called the theoretical distributions. Afterwards the cryptanalyst collects ciphertexts and calculates for every possible value of z_5 and $z_7 \oplus z_3$ the values $(y_1 - z_5) \oplus y_3 \oplus z_7 \oplus z_3$. This gives a set of experimentally measured distributions. If enough ciphertexts are

used the correct values for z_1, z_5 and $z_3 \oplus z_7$ will yield the closest match between a theoretical and an experimental distribution.

The number of required ciphertexts depends on the distribution of the plaintexts. We did experiments with two distributions: real English text and random bytes (with the most significant bit set to zero). The ciphertext requirements can be reduced significantly if the key ranking technique [3] is used: not only the most probable key is given as output, but a small set of key values with high probability to contain the correct value. For the case of English text, using 1000 ciphertexts, $s^R \bmod 8$ can be determined with probability 0.7 and the correct values for one byte of each of z_1, z_5, and $z_3 \oplus z_7$ are in 90% of the cases among the 4 most suggested values (out of 2^{24} possible values). It is reasonable to assume that the probability of successes will be the same for the attacks on the remaining key bytes. Thus, for the case of English text with 1000 ciphertexts we estimate that the correct values of all four bytes of each of z_1, z_5, and $z_3 \oplus z_7$ will be amongst the $4^4 = 256$ most suggested values (out of 2^{96} values) with success probability $0.9^4 \simeq 0.66$. The results are summarized in Table 1. As can be seen the recovery of the keyed rotation works better for random ASCII bytes than for English text, whereas the recovery of the keys z_i works worse. We expect that success probability of the key recovering part of our attack will increase significantly when using more texts.

	# texts	recovering $s^R \bmod 8$	recovering z_i
English text	1000	0.70	0.66
Random bytes	1000	0.90	0.00
English text	5000	0.78	1.00
Random bytes	5000	1.00	0.03

Table 1: Success probability for the ciphertext only attack on Akelarre when the plaintext is known to be English ASCII-coded text; and when the plaintext consists of random bytes, with the most significant bit set to zero.

Subsequently, the approach of the previous section can be used to recover plaintexts. However, note that the attacker must repeat this attack for both sets of values for (v, v_2, w, w_2) of Equation 6.

5 Conclusion

We presented realistic attacks on the block cipher Akelarre, which mixes features of the block cipher IDEA and RC5. Our attacks are independent of the number of rounds used in the cipher and enable the recovery of a limited set of key bits. Once these bits have been found an attacker can obtain the plaintexts of any intercepted ciphertexts, provided that the plaintext space is redundant. Akelarre and our attacks illustrate that mixing the components of two presumably secure block ciphers does not always yield a strong new block cipher.

References

[1] G. Alvarez, D. de la Guiaía, F. Montoya and A. Peinado, "Akelarre: a new block cipher algorithm," *Proceedings of SAC'96, Third Annual Workshop on Selected Areas in Cryptography*, Queen's University, Kingston, Ontario, 1996.

[2] X. Lai, J.L. Massey and S. Murphy, "Markov ciphers and differential cryptanalysis," *Advances in Cryptology, Proceedings Eurocrypt'91, LNCS 547*, D.W. Davies, Ed., Springer-Verlag, 1991, pp. 17–38.

[3] M. Matsui, "The first experimental cryptanalysis of the Data Encryption Standard," *Advances in Cryptology, Proceedings Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 1–11.

[4] R.L. Rivest, "The RC5 encryption algorithm," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 86–96.