Nessie Proposal: NOEKEON

Joan Daemen*, Michaël Peeters*, Gilles Van Assche* and Vincent Rijmen**

*Proton World Intl, Belgium,

{joan.daemen, michael.peeters, gilles.vanassche} @protonworld.com

**ESAT Cosic KULeuven, Belgium

vincent.rijmen@esat.kuleuven.ac.be

Table of Contents

1. Introduction

In this document we describe the cipher NOEKEON. First we present the design rationale and specify the cipher. Subsequently, we motivate the design choices and discuss the resistance of the cipher against known attacks. This is followed by a discussion on the implementation aspects and the references. In the appendix we explain how the propagation analysis was conducted.

Patent Statement: NOEKEON or any of its implementations is not and will not be subject to patents.

2. Design Rationale

NOEKEON is a block cipher with a block and key length of 128 bits. It is an iterated cipher consisting of the repeated application of a simple round transformation, followed by an output transformation. The round transformation includes XORing a *Working Key* to the intermediate result.

The criteria taken into account in the design of NOEKEON are the following:

- **Resistance against cryptanalysis:** NOEKEON is expected to behave as good as can be expected from a block cipher with the given block and key length. There shall be no shortcut attacks.
- **Speed:** on a wide range of platforms;
- **Code/Circuit compactness:** on a wide range of platforms;
- **Resistance against implementation attacks:** the ability to make implementations of the cipher that are resistant against attacks such as timing attacks and power analysis;
- **Design simplicity:** symmetry and simplicity of description.

Obviously, NOEKEON can be used for the protection of confidentiality by applying one of the well-known modes of operation: ECB, CBC, CFB, OFB and filtered counter mode. Moreover, it can be used for the protection of data integrity and authentication if used at the core of a MAC algorithm, as specified in [ISO9797]. Additionally, it can be used as a one-way function, with:

- the input of the one-way function corresponding with the Cipher Key of the block cipher,
- the diversification parameter of the one-way function with the input to the block cipher, and
- the output of the one-way function is the output of the block cipher.

Although NOEKEON is designed to be an all-round block cipher, efficiently and securely implementable on a wide range of platforms, it is particularly well suited for platforms in which resources are scarce, such as Smart Cards. In our opinion, its excellent code/circuit compactness and suitability to be implemented securely with respect to implementation attacks make it the ideal candidate for these platforms.

2.1 Key Length

NOEKEON supports a key length of 128 bits, denoted as “security level: *normal*” in the Nessie call for primitives. Keys of 256 bits (security level: *high*) are currently not supported by NOEKEON, although it is obviously not too difficult to imagine NOEKEON generalisations supporting longer keys.

The underlying reason is that we actually consider the security of a block cipher that provides the level of security implied by a key length of 128 bits to be quite high. Even in the unlikely case that Moore’s law will continue to hold for the next century, there will be no need for having keys longer than 128 bits for about 80 years to protect against exhaustive key search.

2.2 Self-Inverse Bit-Slice Cipher

NOEKEON can be implemented using only bit-wise Boolean operations and (cyclic) shift operations, similarly to 3-WAY [Da95], BASEKING [Da95], and, more recently, the AES proposal Serpent [BiAnKn98.]

The NOEKEON design inherits its large degree of symmetry, its parallelism and simplicity of description from its predecessors 3-WAY and BASEKING.

Moreover, as with 3-WAY and BASEKING, but not with Serpent, the inverse operation of NOEKEON can be executed with the same circuit/program as NOEKEON itself. For cases in which both the cipher and its inverse are needed, this leads to a reduction in code size/circuit area.

2.3 The Key Schedule

As opposed to 3-WAY and BASEKING, NOEKEON does have a key schedule. The key schedule consists of the conversion of the 128-bit *Cipher Key* into a 128-bit *Working Key* that is actually XORed to the intermediate computation result in the round transformation. The function of this key schedule is to protect NOEKEON against related-key attacks [Bi93].

A related-key attack requires cipher executions with different Working Keys that have at least a known and in most cases a carefully chosen relation. The function used to convert the Cipher Key into the Working Key is the cipher itself. The Working Key is the result of applying NOEKEON to the Cipher Key as input and a null-string as Working Key. In dedicated hardware implementations, this has the advantage that the same circuit can be used for the cipher computation and for the key schedule.

For NOEKEON, we define a mode in which the key schedule is not applied, called the *direct-key* mode. This comes down to the fact that the Working Key is the Cipher Key itself. Obviously, as was shown for 3-WAY, this makes the cipher vulnerable for related-key attacks [KeScWa96]. Therefore, we advise to use the direct-key mode only in circumstances where related-key attacks cannot be mounted. The direct-key mode eliminates some of the disadvantages of the presence of the key schedule:

- It takes RAM to store the Working Key and cycles to compute it. If only a few blocks need to be enciphered or MACed, the overhead of the key schedule becomes relevant.
- In actual implementations, the presence of a key schedule may actually expose the key due to attacks that make use of power consumption or radiation [KoJaJu98].

The notorious key diversification mechanism known as *key variants* actually provides the circumstances in which related-key attacks can be mounted. In key variants, a single mother key is diversified into different child keys by XORing different constants to it. If key variants are used, the direct-key mode may not be used. However, most systems using key variants can be easily re-designed resulting in an equally efficient key architecture separating keys based on parameterised one-way functions.

2.4 Implementation Attacks

In the design of NOEKEON, we have taken into account implementation attacks from the beginning. As a matter of fact, in most practical implementations using cryptography, the aspect that is the hardest to protect is the physical protection of the key material. The design of NOEKEON has the following features to enable secure implementations:

- Most of its operations are linear. There are only four non-linear operations in each round and these operate on only two operands each. This allows the application of techniques proposed in [DaPeVa2000] to protect against Differential Power Analysis (DPA). [KoJaJu98]
- In the direct-key mode the Cipher Key is applied directly without prior manipulation. This excludes implementation attacks focusing on the key schedule [BiSh98]. Moreover, key splitting [DaPeVa2000] is straightforward.

3. Specification

NOEKEON is an iterated block cipher with a block and key length of 128 bits.

3.1 The State and the Cipher Key

The Input, Output and Cipher Key used by NOEKEON at its external interface are considered to be one-dimensional arrays of 8-bit bytes numbered upwards from 0 to 15.

The different transformations operate on the intermediate result, called the *State* a, consisting of 4 32-bit words, a[0] to a[3]. The state is initialised with the Input prior to the cipher execution, and the Output is extracted from the State after the cipher execution.

NOEKEON is an iterated block cipher, consisting of the iterated application of a round transformation. The number of rounds is denoted by **Nr**.

3.2 The Round Transformation

The Round Transformation is composed of different transformations. For the specification of the different transformations, we will use a pseudo C notation. We have:

```
Round(Key,State,Constant1,Constant2) {
      State[0] ^= Constant1;
      Theta(Key,State);
      State[0] ^= Constant2;
      Pi1(State);
      Gamma(State);
      Pi2(State);
}
```

In our notation, the “functions” (`Round`, `Theta`, `Pi1`, `Gamma`, `Pi2`, …) operate on the State to which a pointer (`State`) is provided.

Of the two constants that are added, there is always one set to zero. If the round transformation is used in the cipher, Constant2 is 0, if the round transformation is used in the inverse cipher, Constant1 is 0.

The component transformations are specified in the following subsections.

3.3 Gamma

Gamma is an involutive non-linear mapping that operates on the state in the following way:

```
Gamma(a){
      a[1] ^= ~a[3]&~a[2];
      a[0] ^=  a[2]& a[1];

      tmp = a[3]; a[3] = a[0]; a[0] = tmp;
      a[2] ^= a[0]^a[1]^a[3];

      a[1] ^= ~a[3]&~a[2];
      a[0] ^=  a[2]& a[1];
}
```

Note: the nonlinear operation `~a[3]&~a[2]` can be replaced by the equivalent expression `~(a[3]|a[2])`.

Gamma operates independently on 32 4-tuples of bits, called *boxes*. These boxes consist of the 4 bits in each word that are in the same position. For example, box 9 consists of bit 9 of a[0], bit 9 of a[1], bit 9 of a[2] and bit 9 of a[3]. The *value* of box 9 can be represented by a hexadecimal digit representing the four bits in the order 3, 2, 1, 0.

Gamma can be specified alternatively as an S-box applied to each of the boxes of the state. This S-box specification is given in Table 1.

Input:	$a_3a_2a_1a_0$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output:	$b_3b_2b_1b_0$	7	A	2	C	4	8	F	0	5	9	1	E	3	D	B	6

Table 1 Gamma S-box specification

3.4 Theta

Theta is a linear mapping that takes the Working Key *k* and operates on the state *a* in the following way:

```
Theta(k,a){
      temp = a[0]^a[2]; temp ^= temp>>>8 ^ temp<<<8;
      a[1] ^= temp;
      a[3] ^= temp;

      a[0] ^= k[0]; a[1] ^= k[1]; a[2] ^= k[2]; a[3] ^= k[3];

      temp = a[1]^a[3]; temp ^= temp>>>8 ^ temp<<<8;
      a[0] ^= temp;
      a[2] ^= temp;
}
```

The inverse of Theta is given by applying the three involutive mappings in reverse order. Using the algebraic property of linear mapping L: L(a+k) = L(a) + L(k) and the fact that the first and the last component mapping of Theta commute, the inverse of Theta can be expressed as:

```
Theta(k',a)
```

With k' the Working Key after being processed by Theta with an all-zero Working Key (NullVector):

```
Theta(NullVector,k)
```

In other words, the inverse of Theta is equal to Theta itself with another Working Key value k'. The key k' is obtained by applying Theta to the Working Key.

3.5 The Shift Operations

The shift operations Pi1 and Pi2 perform cyclic shifts of three of the four words over different offsets. We have:

```
Pi1(a){ a[1] <<<= 1; a[2] <<<= 5; a[3] <<<= 2;}
Pi2(a){ a[1] >>>= 1; a[2] >>>= 5; a[3] >>>= 2;}
```

Pi1 and Pi2 are each other's inverses.

3.6 The Cipher

The cipher consists of a number of rounds, followed by an additional application of Theta:

```
Noekeon(WorkingKey,State) {
      For( i=0 ; i<Nr ; i++) Round(WorkingKey,State,Roundct[i],0);
      State[0] ^= Roundct[Nr];
      Theta(WorkingKey,State);
}
```

The inverse of the cipher is:

```
InverseNoekeon(WorkingKey,State) {
      Theta(NullVector,WorkingKey);
      For( i=Nr ; i>0 ; i--) Round(WorkingKey,State,0,Roundct[i]);
      Theta(WorkingKey,State);
      State[0] ^= Roundct[0];
}
```

The number of rounds **Nr** is 16.

The only difference between NOEKEON and its inverse is in the computation of the Working Key for the inverse operation and the application of the round constants.

In the *indirect-key* mode the Working Key is derived from the Cipher Key by applying the NOEKEON with an all-zero Working Key ("NullVector"):

```
WorkingKey = CipherKey;
Noekeon(NullVector,WorkingKey);
```

In the *direct-key* mode the Working Key is equal to the Cipher Key:

```
WorkingKey = CipherKey;
```

Note that the Working Key is not varied between the rounds, in both the direct-key mode and the indirect-key mode.

3.7 The Round Constants

The round constants are of the form:

```
Roundct[i] = ('00','00','00',RC[i])
```

Hence, only the least significant byte differs from 0.

The round constants RC[1] to RC[Nr] can be computed in a recursive way:

```
RC[0] = 0x80;
if (RC[i]&0x80 != 0) RC[i+1] = RC[i]<<1 ^ 0x1B else RC[i+1] = RC[i]<<1;
```

This corresponds with a linear feedback shift register. If desired, the round constants can be computed in reversed order:

```
if (RC[i]&0x01 != 0) RC[i-1] = RC[i]>>1 ^ 0x8D else RC[i-1] = RC[i]>>1;
```

Note: the round constants are computed in the same way as in Rijndael [DaRi99], with the exception of the initial value RC[0].

4. Motivation of Design Choices

In the following subsections, we will motivate the choice of the specific transformations and constants.

4.1 Theta

The design criteria for Theta were the following:

1. Involution (if the key is 0): to be able to use the same round transformation both in NOEKEON and its inverse.
2. Small number of operations;
3. Relevant diffusion power;
4. Symmetry;
5. Simplicity of description.

Theta has been constructed in the following way:

1. We chose a simple involutive mapping that modifies two words by XORing a linear function of the XOR of the other two words. This linear function consists of taking a word X, rotating it over a byte to the left to give Y and rotating it over a byte to the right to give Z and XORing X, Y and Z. This mapping takes 5 XOR operations and two rotations (that disappear in implementations on 8-bit processors).
2. Theta consists of applying the described mapping, where the state words in odd positions are modified, followed by XORing the key to the state (taking 4 XOR operations), followed by again applying the described mapping, where the state words in even positions are modified.

Theta operates independently on 8 16-tuples of bits, called *columns*. These columns consist of the 4 sets of 4 bits in the state words in positions that are equal modulo 8. For example, *column* 3 consists of bits 3, 11, 19, 27 of a[0], bits 3, 11, 19, 27 of a[1], bits 3, 11, 19, 27 of a[2] and bits 3, 11, 19, 27 of a[3].

Table 2 is the input-output Hamming weight distribution table of Theta. It gives the distribution of the 2^{16} input-output couples for one column over the input Hamming weight and the output Hamming weight. With the exception of 8 inputs with a Hamming weight of 2 that give an output of Hamming weight of 2, the sum of Hamming weights of input and output is at least 8.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1							16								
2		8				48				48				16	
3					112				448						
4				348				1152				224			
5			112				3136				1120				
6		48				3928				3984				48	
7	16				3136				7840				448		
8				1152				10556				1152			
9			448				7840				3136				16
10		48				3984				3928				48	
11					1120				3136				112		
12				224				1152				348			
13							448				112				
14		16				48				48				8	
15									16						

Table 2 Input-output Hamming weight distribution table of Theta

4.2 The Shift Offsets

The shift offsets of Pi1 and Pi2 have been selected according to the following criteria:

1. Pi2 is the inverse of Pi1: to be able to use the same round transformation both in NOEKEON and its inverse.
2. The four offsets of Pi1 are different modulo 8;
3. The offset for word 0 is 0;
4. The array of offsets is selected from the set of arrays that optimise diffusion over a single round according to a simple criterion. From the resulting set of arrays, the first one, in lexicographical order, was selected.

The diffusion measure is the number of boxes affected at the output of the linear part of the round transformation by a single box at its input. The linear part of the round transformation is (Pi1 ° Theta ° Pi2). In other words, the measure counts the number of nonzero boxes in an output vector (four 32-bit words) corresponding to an input vector (four 32-bit words) with a single all-one box.

Thanks to the symmetry in these three transformations, the number of nonzero S-boxes in the output vector is independent of the position of the non-zero box in the input. Moreover, if this is applied to the inverse operation, the same result is obtained. For the current choice of shift offsets, the number of active S-boxes is 23. This is the best result achievable, under criteria 1-3.

4.3 Gamma

The design criteria for Gamma are:

1. Involution: to be able to use the same round transformation both in NOEKEON and its inverse;
2. Low number of bit-wise Boolean operations;
3. Maximum non-trivial input-output correlation is ½;
4. Maximum difference propagation probability is ¼.

Gamma has been constructed in the following way:

1. We chose a simple non-linear involutive mapping, that modifies word 1 by XORing the NOR of words 3 and 2 and word 0 by XORing the AND of words 2 and 1. The result is independent of the order in which these operations are executed. This mapping takes two XOR, one AND and one NOR operation.
2. We looked in the range of mappings that consist of the application of the selected non-linear mapping twice, with in between a linear involutive mapping. This results in an involutive mapping Gamma.
3. The 64 mappings that satisfy criteria 3 and 4 stated above have been subject to difference and correlation propagation analysis.
4. The mapping that performed best in our propagation analysis was selected. The selected linear mapping in between takes three XORs and one word permutation (that can be implemented implicitly).

Table 3 lists the (scaled) difference propagation probabilities (prop ratios) of Gamma applied to a single box. Table 4 lists the (scaled) input-output correlation values of Gamma.

	0	1	2	4	8	3	5	6	9	A	C	7	B	D	E	F
0	16		.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	.	.	.	.	.	.	.	.	.	4	.	.	4	4	4
2	.	.	.	2	4	.	2	2	.	.	.	2	4	.	.	.
4	.	.	2	2	2	2	.	2	.	2	2	.	.	2	.	.
8	.	.	4	2	.	4	2	2	.	.	.	2	.	.	.	.
3	.	.	.	2	4	.	2	2	.	.	.	2	4	.	.	.
5	.	.	2	.	2	2	2	.	.	2	.	2	.	.	2	2
6	.	.	2	2	2	2	.	2	.	2	.	.	.	.	2	2
9	.	.	.	.	.	.	.	.	4	.	2	.	4	2	2	2
A	.	.	.	2	.	.	2	2	.	.	2	2	.	2	2	2
C	.	4	.	2	.	.	.	.	2	2	.	2	.	.	4	.
7	.	.	2	.	2	2	2	.	.	2	2	2	.	2	.	.
B	.	.	4	.	.	4	.	.	4	.	.	.	4	.	.	.
D	.	4	.	2	.	.	.	.	2	2	.	2	.	4	.	.
E	.	4	.	.	.	.	2	2	2	2	4	.	.	.	.	.
F	.	4	.	.	.	.	2	2	2	2	.	.	.	.	.	4

Table 3 Difference propagation probabilities (prop ratios) of the Gamma S-box.

	0	1	2	4	8	3	5	6	9	A	C	7	B	D	E	F
0	16	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
1	.	–8	–4	.	8	4	.	–4	.	4	.	–4	–4	.	–4	–4
2	.	–4	4	.	.	.	4	–4	–4	–4	–8	.	8	–4	–4	.
4	.	.	.	.	.	–8	.	.	–8	.	.	–8	.	8	.	.
8	.	8	.	.	.	8	8	.	.	.	.	–8	.	.	.	.
3	.	4	.	–8	8	4	–4	.	–4	.	.	4	4	4	.	4
5	.	.	4	.	8	–4	8	4	.	–4	.	4	–4	.	4	–4
6	.	–4	–4	.	.	.	4	4	4	4	–8	.	.	4	4	8
9	.	.	–4	–8	.	–4	.	4	8	–4	.	–4	4	.	–4	–4
A	.	4	–4	.	.	.	–4	4	–4	4	–8	.	.	–4	4	–8
C	.	.	–8	.	.	.	.	–8	.	–8	.	.	.	.	8	.
7	.	–4	.	–8	–8	4	4	.	–4	.	.	4	–4	4	.	–4
B	.	–4	8	.	.	4	–4	.	4	.	.	–4	4	4	8	–4
D	.	.	–4	8	.	4	.	4	.	–4	.	4	4	8	–4	–4
E	.	–4	–4	.	.	.	4	4	–4	4	8	.	8	–4	4	.
F	.	–4	.	.	.	4	–4	8	–4	–8	.	–4	–4	–4	.	4

Table 4 Input-Output Correlations in the Gamma S-box.

4.4 The Round Constants

The round constants are added to eliminate the following symmetry properties:

- **In the round transformation**: the round transformation processes the different boxes in the same way;
- **Between the rounds**: the round transformation is the same for all rounds.

4.5 Number of Rounds

In the design of NOEKEON, the number of rounds can easily be varied. We estimate that 16 rounds is more than sufficient to thwart any shortcut attacks. (A shortcut attack is an attack more efficient than exhaustive key search.)

The best theoretical attack known to us is a linear cryptanalysis attack making use of an iterative linear trail of period 2 rounds, having a correlation coefficient of 2^{-14} per two rounds. This can be stretched to 9 rounds, giving a correlation coefficient equal to 2^{-62}. In the assumption that one round may be skipped at the beginning and two at the end, this would lead to an attack on a 12-round version of NOEKEON. Taking 16 rounds gives NOEKEON a security margin of 4 rounds or 33 %.

4.6 Trapdoors and Hidden Weaknesses

We formally state that we have not inserted any trapdoor or any hidden weakness in NOEKEON.

Moreover, we believe that the cipher structure does not offer enough degrees of freedom to hide a trapdoor or any other weakness.

5. Strength against Known Attacks

5.1 Differential and Linear Cryptanalysis

Differential cryptanalysis (DC) of DES was first successfully performed by Eli Biham and Adi Shamir [BiSh91]. Linear cryptanalysis (LC) of DES was first successfully performed by Mitsuru Matsui [Ma94]. We will use the formalism of [Da95] to describe the LC and DC properties. Chapter 5 of [Da95] was included in Annex of the AES proposal Rijndael [DaRi99] and can be easily obtained in PDF format. We refer to it as [Da98].

5.1.1 Differential Cryptanalysis

For a DC attack to exist, there must be a predictable difference propagation:

- over all but a few (typically 2 or 3) rounds;
- with a prop ratio (the relative amount of all input pairs that for the given input difference give rise to the output difference) significantly larger than 2^{-127}.

A difference propagation from a difference pattern *a*' to a difference pattern *b*' is *composed* of differential trails, where its prop ratio is the sum of the prop ratios of all differential trails that have *a*' as initial difference and *b*' as final difference pattern. To be resistant against DC, it is therefore a necessary condition that there are no differential trails with a predicted prop ratio higher than 2^{-127}.

For NOEKEON, we can guarantee that there are no 4-round differential trails with a predicted prop ratio above 2^{-48}. For the significance of these predicted prop ratios, we refer to [Da98].

In [LaMaMu91] it has been proposed to perform differential cryptanalysis with another notion of difference. This is especially applicable to ciphers where the key addition is not a simple XOR operation. Although in NOEKEON the keys are applied using XORs, it was investigated whether attacks could be mounted using another notion of difference. We have found no attack strategies better than using XOR as the difference.

5.1.2 Linear Cryptanalysis

LC attacks are possible if there are predictable input-output correlation values over all but a few (typically 2 or 3) rounds significantly larger than 2^{-64}. An input-output correlation is *composed* of linear trails, where its correlation is the sum of the correlation coefficients of all linear trails that have the specified initial and final selection patterns. The correlation coefficients of the linear trails are signed and their sign depends on the value of bits of the Working Key. To be resistant against LC, it is a necessary condition that there are no linear trails with a correlation coefficient higher than 2^{-64}.

For NOEKEON, we can guarantee that there are no 4-round linear trails with a correlation coefficient above 2^{-24}. For the significance of these correlation coefficients, we refer to [Da98].

5.1.3 Weight of Differential and Linear Trails

In [Da98], it is shown that:

- The *prop ratio* of a *differential trail* can be approximated by the product of the prop ratios of its active S-boxes.
- The *correlation* of a *linear trail* can be approximated by the product of input-output correlations of its active S-boxes.

The *wide trail strategy* [Da98] can be summarised as follows:

- Choose an S-box where the maximum prop ratio and the maximum input-output correlation are as small as possible. For the Gamma S-box this is respectively ¼ and ½.
- Construct the diffusion layer in such a way that there are no multiple-round trails with few active S-boxes.

The guarantees mentioned above are obtained in the following way:

- We have searched the complete space of 4-round trails (both linear and differential) with less than 24 active S-boxes,
- For all 4-round linear traces with less than 24 active S-boxes, we have verified that the correlation coefficient is at most 2^{-24}.
- For all 4-round differential traces with less than 24 active S-boxes, we have verified that the prop ratio is at most 2^{-48}.
- For a description of the propagation analysis performed, we refer to Appendix A.

5.1.4 Results of the Propagation Analysis

There are 6 four-round linear trails with 21 active S-boxes, 33 trails with 22 active S-boxes and 88 trails with 23 active S-boxes. None of them have a correlation coefficient above 2^{-24}.

There are 2 four-round differential trail with 22 active S-boxes and 12 trails with 23 active S-boxes. None of them has a prop ratio above 2^{-48}.

In fact, due to symmetry properties within a round, each of the trails found represents a class of 32 equivalent trails.

For most of the trails found, adding rounds leads to an enormous increase of the correlation coefficient/prop ratio. However, among the linear trails with 22 active S-boxes we have identified four iterative 2-round trails. Due to the symmetry properties, this adds up to 128 different iterative trails. All these trails have 11 active S-boxes per two rounds and the correlation coefficient, computed using the input-output correlation table of Gamma, is 2^{-6} for the even rounds and 2^{-8} for the odd rounds. In theory, this allows attacks against NOEKEON using a 9-round linear trail, requiring at least 2^{124} known plaintext-ciphertext pairs.

Although the existence of these trails imposes an upper limit to the amount of resistance against LC that can be achieved per round, the absolute value of this upper limit, 2^{-7} per round, is quite high. In DES for example, the upper limit imposed by an iterative linear trail is $2^{-1.5}$. One might say that one round of NOEKEON provides more protection against LC than 4 DES rounds.

Note: The iterative trails found are of the form $(a \rightarrow A \mid A \rightarrow a)$. Their presence is the consequence of the self-inverse structure of the cipher (see Appendix for the notation):

- Lambda is an involution, hence if there is a step $(a \rightarrow A)$, there is also a step $(A \rightarrow a)$.
- If we can find a step $(a \rightarrow A)$ for which both a and A are self-compatible through Gamma, the two-round iterative trails $(a \rightarrow A \mid A \rightarrow a)$ exists. In this respect, the nonzero elements on the main diagonal of the propagation tables of the S-box are relevant.

5.1.5 Clustering of Differential Trails and Linear Trails

In our propagation analysis, we have not seen indications of clustering of differential trails resulting in high probability differentials or of linear trails leading to high input-output correlation values. Based on the bit-orientation of the cipher, we do not expect this to occur outside of the investigated part of the trails space either.

Moreover, we do not expect the symmetry properties of the cipher to be exploited in differential or linear cryptanalysis.

5.2 Symmetry Properties and Weak Keys as in DES

Despite the large amount of symmetry, care has been taken to eliminate symmetry in the behaviour of the cipher (e.g., the complementation property in DES). This is obtained by the round constants that are different for each round. The fact that the cipher and its inverse have different round constants practically eliminates the possibility for weak and semi-weak keys, as existing for DES.

So-called Slide Attacks [BiWa99] are applicable to iterative ciphers in which the round transformation is the same for each round. This would be the case for NOEKEON if the round transformation did not include the addition of a round constant. We rely on these constants that are different for each round, to prevent slide attacks.

5.3 Truncated Differentials

The concept of truncated differentials was first published by Lars Knudsen [Kn95]. The corresponding class of attacks exploit the fact that in some ciphers differential trails tend to *cluster* [Da98]. Clustering takes place if for certain sets of input difference patterns and output difference patterns, the number of differential trails is exceedingly large. The expected probability that a differential trail stays within the boundaries of the cluster can be computed independently of the prop ratios of the individual differential trails. For NOEKEON, we do not expect truncated differentials to pose a risk thanks to the bit-oriented nature of the cipher.

5.4 Interpolation Attacks

In [JaKn97] Jakobsen and Knudsen introduced a new type of attack on block ciphers. In these attacks, the attacker constructs polynomials using cipher input/output pairs. This attack is feasible if the components in the cipher have a compact algebraic expression and can be combined to give expressions with manageable complexity. The basis of the attack is that if the constructed polynomials (or rational expressions) have a small degree, only few cipher input/output pairs are necessary to solve for the (key-dependent) coefficients of the polynomial.

Assessing the security of a block cipher against interpolation attacks is a difficult task. Firstly, there is a large variety of descriptions to consider. A component with a complicated polynomial description A(x), can possibly be described very efficiently as a rational form $P(x)/Q(x)$, with P(x) and Q(x) very compact polynomials. Since there is no theory available on the minimal complexity of the representation of components of the sizes typically employed in block ciphers, all a cryptanalyst can do, is to try out a number of representations.

Secondly, the complexity of the description of a mapping over a finite field depends on the basis chosen to represent the field. Circuit designers often use this fact to reduce the number of gates required to implement a given mathematical operation. A well-known example is the mapping $x \Rightarrow x^2$ over a finite field of characteristic 2. In a representation using a normal basis, this mapping corresponds to a simple rotation. When cryptanalyzing a block cipher, we are usually facing the opposite problem: the logical operations on the bit level are known, but we are looking for a (compact) mathematical description.

For the specific case of NOEKEON, we note the following:

- The dimensions of the used components seem to suggest to consider mappings operating on variables coded in 32, 16, 8, 4, 2 or 1 bits.
- In a Galois field with normal basis representation, a rotation over n bits corresponds to the mapping $x \Rightarrow x^{2^n}$. The same is true if the bits represent elements in the ring of polynomials modulo $x^m - 1$, where m is the number of bits of the variables.
- Bit-wise XOR corresponds to addition in all Galois fields with even characteristic, and in the rings of polynomials with binary coefficients.

We have experimented with different word lengths to give simple expressions of the round transformation. For 1-bit and 2-bit variables, the expressions are reasonably simple but contain many terms. This gives rise to an explosion of the number of terms if the expression is used to describe multiple rounds. If variables are used consisting of more bits, the number of variables in the expression diminishes. However, the AND and NOR operations in Gamma lead to complex expressions containing many terms, practically limiting the usability of the expression to describe multiple rounds. From our experiments we conclude that it is highly unlikely that NOEKEON will be subject to an attack that uses the manipulation of algebraic expressions.

5.5 Weak Keys as in IDEA

The weak keys discussed in this subsection are keys that result in a block cipher mapping with detectable weaknesses. The best-known case of weak keys is that of IDEA [Da95]. Typically, this weakness occurs for ciphers in which the non-linear operations depend on the actual key value. This is not the case for NOEKEON, where keys are applied using the XOR and all non-linearity is in Gamma. In NOEKEON, there is no restriction on key selection.

5.6 Related-Key Attacks

In [Bi96], Eli Biham introduced a related-key attack. Later John Kelsey, Bruce Schneier and David Wagner demonstrated that several ciphers, including 3-WAY and BASEKING are prone to related-key attacks in [KeScWa96].

To protect against related-key attacks, NOEKEON has a key schedule that basically consists of applying the cipher to the Cipher Key as input and using the output of this operation as Working Key. We expect this to make known relations between Cipher Keys unusable. Moreover, thanks to the bijective nature of the block cipher transformation, colliding Cipher Keys giving rise to the same Working Key cannot occur. Moreover, the key schedule has the advantage that it can be performed with the same circuit/code that is used for the cipher itself.

However, the key schedule in NOEKEON is optional and we think that related-key attacks can easily be prevented by intelligently designing the protocol in which the block cipher is used. In circumstances in which a related-key cannot be mounted, we advise the usage of NOEKEON in direct-key mode.

6. Implementation Aspects

NOEKEON is suited to be implemented efficiently on 32-bit processors and in dedicated hardware. However, it is also reasonably efficient on 8-bit processors.

6.1 Flexibility and Parallelism

At first sight, the round transformation of NOEKEON has a sequential nature, with its three stages in Gamma, its three stages in Theta and the two rotation steps Pi1 and Pi2. However, the linearity of most of the transformations allows a lot of flexibility in their implementation:

- The key addition, performed in the middle of Theta, can be moved to the beginning of Theta, before Pi2, after Theta or even after Pi1 at the cost of the calculation of an equivalent Working Key, that has to be done only once for each Cipher Key usage.
- The first and last steps of Theta can be done simultaneously and the rotations can be integrated in this process.

6.2 Hardware Suitability

The fact that NOEKEON consists of the repeated application of a reasonably simple round transformation and the fact that the key schedule makes use of the same circuit, makes that it can be implemented in a remarkably small number of gates. Moreover, the inverse operation of NOEKEON can be performed with the same circuit.

The gate count of the round transformation is 640 XOR, 64 AND and 64 NOR (two-input) gates:

- Theta: 416 XOR gates
  - First part of Theta: 32+16+32+32+32: 144 XOR gates
  - Key addition: 128 XOR gates
  - Last part of Theta: 144 XOR gates
- Round constant addition and computation (2 LFSRs): less than 32 XOR gates
- Pi1 and Pi2: none
- Gamma: 224 XORs, 64 AND and 64 NOR gates
  - First part of Gamma: 32 AND, 32 NOR and 64 XOR gates
  - Middle part of Gamma: 96 XOR gates
  - Last part of Gamma: 32 AND, 32 NOR and 64 XOR gates

The additional Theta that forms the output transformation can be implemented by explicitly hardwiring it or by feeding the intermediate result between the application of Theta and Pi1 to the output. In both cases, this “hardware Theta” can be used for the computation of the key required for the inverse cipher.

In a hardware circuit, the number of gates in the critical path determines the clock speed. Theta can be implemented with a depth of three XOR gates, Gamma can be done with a depth of four XOR gates and one AND (or NOR) gate. Pi1 and Pi2 can be hardwired, resulting in a gate delay of 7 XOR gates, 1 AND gate and probably a multiplexer to allow the loading of the input.

The fact that the circuit is so compact leaves more room for possible additional circuitry to protect against implementation attacks.

6.3 Software Suitability

6.3.1 8-bit Processors

All bit-wise Boolean operations that operate on 32-bit words can be easily split in four operations on 8-bit words. The cyclic shifts of 32-bit words can be implemented by the combination of shift operations of the 8-bit words and bit-wise Boolean operations. The cyclic shift operations in Theta do not require explicit coding on an 8-bit processor, hence only the 6 rotations of Pi1 and Pi2 have to be implemented. Thanks to the fact that the round transformation is the same for each round and that the key schedule uses the same round transformation, the code size is very limited.

An implementation of the direct-key mode requires only 16 bytes to keep the state, one byte to keep track of the round number and a few bytes (say 3) for keeping temporary variables. If the indirect-key mode is implemented, 16 more bytes are needed to store the Working Key.

6.3.2 32-bit Processors

6.3.2.1 ARM7

We have programmed NOEKEON in *direct-key* mode on the ARM7 RISC microprocessor using the ARM Software Development Toolkit from Advanced RISC Machine Ltd.

The ARM processor is particularly well suited for an implementation of NOEKEON. Indeed it has an orthogonal instruction set and all binary operations accept one target register and two source registers, of which one can be rotated in the same cycle. Moreover there are a lot of work registers available. Consequently, this allows a very compact and efficient implementation of a round where Pi1 and Pi2 are merged in the computation of either Gamma or Theta. We also move the application of the key at the beginning of Theta, and hence both encryption and decryption processes have the same execution time.

Table 5 shows the statistics obtained for a straightforward implementation and two optimised implementations of NOEKEON in *direct-key* mode in ARM7 assembly language.

	code size	NOEKEON	NOEKEON^{-1}	bit rate @ 28.56MHz
Straightforward	428 bytes	1247 cycles	1271 cycles	2.9 Mbit/s
Optimised for size	332 bytes	712 cycles	712 cycles	5.1 Mbit/s
Optimised for speed	3688 bytes	475 cycles	475 cycles	7.7 Mbit/s

Table 5: Implementation statistics for NOEKEON in direct-key mode in ARM7 assembler

For NOEKEON in *indirect-key* mode, the key set-up time can be estimated as being equal to encryption time. The last column gives the bit rate assuming a frequency at 28.56MHz obtained from the standard frequency of 3.57MHz for smart cards with a clock multiplier.

Regarding the RAM usage, if both the state and the key are forwarded through registers, both optimised implementations do not need any byte of memory.

6.3.2.2 Pentium

We programmed NOEKEON in *direct-key* mode on the Intel Pentium II processor using Microsoft Visual C/C++ 6.0 under Windows NT 4.0.

Table 6 shows the statistics obtained for an early implementation in assembly language using an interface adapted to little endianness and applying only basic rules to increase intrinsic parallelism. Further optimisations are probably possible (e.g. using the MMX instruction set) but were not investigated here. These statistics were collected on a 500Mhz-Pentium II processor, but the speed figures are shown for the Pentium 200MHz reference platform.

	NOEKEON	NOEKEON^{-1}	bit rate @ 200MHz
Pentium II – 200 MHz	525 cycles	525 cycles	49 Mbit/s

Table 6: Implementation statistics for NOEKEON in Direct-key mode for the Pentium II

In the case of the indirect-key mode, the performance of the key-setup is similar to those of the encryption.

6.4 Protection against Implementation Attacks

In the design of the cipher, we have taken into account the fact that it must be feasible to implement it in such a way that it provides protection against implementation attacks such as timing attacks and power analysis attacks.

One can protect against timing attacks and simple power analysis by coding the cipher as a fixed sequence of instructions. As NOEKEON only consists of a fixed sequence of simple operations, this is straightforward.

Protection against Differential Power Analysis (DPA) [KoJaJu98] can be obtained by applying several mechanisms, preferably at the same time. One of the mechanisms is called *state splitting.* This method was proposed in [GoPa99, CJRR99] and consists in splitting the state in two parts (one of which is supposed to be random) that give XORed back the actual state value. If properly implemented, this method eliminates any correlation between values inside the machine and any intermediate computation result of the cipher, eliminating the possibility for first-order DPA. We have shown how this can be applied to BASEKING in [DaPeVa2000]. It turns out that for the linear operations the computations can be done independently on the two parts of the state. An additional overhead is in the non-linear operations, in which operations must be performed using parts of both split states.

The mechanisms described in [DaPeVa2000] apply equally well to the non-linear parts in NOEKEON, that are in fact of the same type: the addition of an AND (or NOR) of two words to a third word by means of an XOR. The number of these operations is only 4 per round, limiting the additional overhead to a minimum.

We programmed an anti-DPA version of NOEKEON in *direct-key* mode that implements the discussed mechanism on the ARM7 RISC microprocessor. We did not optimise this implementation as extensively as we did in Section 6.3.2.1. So, in order to have a meaningful comparison of the implementation cost, Table 7 compares the results obtained for this version with the straightforward implementation of NOEKEON given in Section 6.3.2.1.

	code size	NOEKEON	NOEKEON^{-1}	bit rate @ 28.56MHz
Straightforward	428 bytes	1247 cycles	1271 cycles	2.9 Mbit/s
Anti-DPA	792 bytes	2145 cycles	2254 cycles	1.7 Mbit/s

Table 7: Implementation cost of the anti-DPA version of NOEKEON in direct-key mode in ARM7 assembler

We see that the implementation cost for the anti-DPA version is 364 bytes for the code size, and an average of 940 cycles for the execution time. Although this almost doubles the cost of the classical implementation, we find this a very attractive solution to counter DPA attacks.

The mechanisms mentioned above can be applied several times, say *n* times, in order to counter higher-order DPA attacks. For small *n*, the implementation cost grows approximately linearly as a function of *n*.

7. Strenghs and Advantages of NOEKEON

Compared to other block ciphers, NOEKEON has a number of strengths and advantages that make it quite unique.

NOEKEON:

- Is ultra compact and fast in dedicated hardware implementations;
- Allows efficient DPA-resistant software implementations;
- Has very low RAM requirements in software implementations;
- Has compact code in software implementations;
- Runs efficiently on a wide range of platforms;
- Has a design that is so clean and simple that it can be memorised by an average person.

8. References

[Bi93] E. Biham, *New types of cryptanalytic attacks using related keys*, in Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765, T. Helleseth, Ed., Springer-Verlag, 1993, pp. 398-409.

[BiAnKn98] E. Biham, R. Anderson and L. Knudsen, *AES proposal Serpent*, 1998.

[BiSh91] E. Biham and A. Shamir, *Differential cryptanalysis of DES-like cryptosystems*, in Journal of Cryptology, Vol. 4, No. 1, 1991, pp. 3-72.

[BiSh98] E. Biham and A.Shamir, *Power analysis of the key scheduling of the AES candidates*. In 2nd AES Candidates Conference, March 1999.

[BiWa99] A. Biryukov and D. Wagner, *Slide Attacks*. In Fast Software Encryption, LNCS 1636, L. Knudsen, Ed., Springer—Verlag 1999, pp. 245—259.

[CJRR 99-1] S. Chari, C. Jutla, J. Rao and P. Rohatgi. *A Cautionary note regarding evaluation of AES candiates on Smart-cards*, in Proceedings of the 2nd AES Candidates Conference, March 1999.

[CJRR 99-2] S. Chari, C. Jutla, J. Rao and P. Rohatgi. *Towards sound approaches to counteract power-analysis attacks*, in Advances in Cryptology – Crypto '99, pages 398–412. Springer-Verlag, 1999.

[Da95] J. Daemen, *Cipher and hash function design strategies based on linear and differential cryptanalysis*, Doctoral Dissertation, March 1995, K.U.Leuven.

[Da98] J. Daemen, *Propagation and Correlation,* Chapter 5 of *Cipher and Hash Function Design*, distributed as PDF file as annex to AES submission Rijndael.

[DaPeVa2000] J. Daemen, M. Peeters and G. Van Assche, *Bitslice ciphers and Power Analysis Attacks*, in Fast Software Encryption 2000, Springer-Verlag 2000.

[DaRi99] J. Daemen and V. Rijmen, *AES Proposal: Rijndael*, AES algorithm submission, September 3, 1999, available at http://www.nist.gov/aes.

[GoPa99] L. Goubin and J. Patarin, *DES and differential power analysis*, in CHES '99, volume 1717, pages 158 —172. Springer-Verlag 1999.

[ISO9797] ISO/IEC 9797, *Information technology - security techniques - data integrity mechanism using a cryptographic check function employing a block cipher algorithm*, International Organization for Standardization, Geneva, 1994 (second edition).

[JaKn97] T. Jakobsen and L.R. Knudsen, *The interpolation attack on block ciphers*, in Fast Software Encryption, LNCS 1267, E. Biham, Ed., Springer-Verlag, 1997, pp. 28-40.

[KeScWa96] J. Kelsey, B. Schneier and D. Wagner, *Key-schedule cryptanalysis of IDEA, GDES, GOST, SAFER, and Triple-DES*, in Advances in Cryptology, Proceedings Crypto '96, LNCS 1109, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 237-252.

[Kn95] L.R. Knudsen, *Truncated and higher order differentials,* in Fast Software Encryption, LNCS 1008, B. Preneel, Ed., Springer-Verlag, 1995, pp. 196-211.

[KoJaJu98] P. Kocher, J. Jaffe and B. Jun. *Introduction to differential power analysis and related attacks*. http://www.cryptography.com/dpa/technical/index.html.

[LaMaMu91] 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.

[Ma94] M. Matsui, *Linear cryptanalysis method for DES cipher,* Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 386-397.

9. Appendix A: Propagation Analysis

9.1 Preliminaries

Difference patterns (DC) and selection patterns (LC) can be seen as propagating through the transformations of the different rounds of the block cipher to form linear and differential trails.

For the study of the propagation of these patterns, the round transformation splits naturally in two parts: the linear part Lambda = (Pi1 ° Theta ° Pi2) and the non-linear part Gamma. For the propagation of the difference and selection patterns through different rounds we use the following convention:

$$a \xrightarrow{\text{Lambda}} A \overset{\text{Gamma}}{|} b \xrightarrow{\text{Lambda}} B \overset{\text{Gamma}}{|} c \xrightarrow{\text{Lambda}} C \overset{\text{Gamma}}{|} d$$

The propagation through Lambda is denoted by ($a \rightarrow A$), also called a *step*. Because of the linearity of Lambda it is fully deterministic: both for LC and DC patterns, we have:

$$A = \text{Lambda}(a).$$

The fact that the relation is the same for LC and DC is thanks to the fact that the Lambda is an orthogonal function. If represented in a matrix, its inverse is its transpose.

The propagation through Gamma is more complicated. One can say that a pattern A and a pattern b are *compatible* or *incompatible*.

- For difference patterns, compatibility means that there is non-zero probability that the difference pattern A may propagate to the difference pattern b through Gamma.
- For selection vectors, compatibility means that there is a non-zero correlation between the parity of the bits indicated by the selection pattern A at the input of Gamma and the parity of the bits indicated by the selection pattern b at the output of Gamma.

If A and b are compatible, this is denoted by $A \mid b$. One step ($b \rightarrow B$) is compatible with another step ($a \rightarrow A$) if $A \mid b$, i.e., they can be chained to form a trail ($a \rightarrow A \mid b \rightarrow B$).

Compatibility between patterns A and b requires the following:

- The positions of the active S-boxes (and hence also their number) are the same in A and b.
- For each of the active S-boxes, there must be nonzero entry in the input-output difference propagation table (Table 3) or input-output correlation table (Table 4) in the entry corresponding to the input and output S-box values.

The number of active S-boxes in a pattern a is denoted by $\#a$ and also called the *box weight*. This number is equal to the Hamming weight of the bit-wise OR of the four words of a pattern. It follows that if $A \mid b$, they have the same number of active S-boxes:

Both for DC and LC, we define box weight of a step ($a \rightarrow A$) as the number of nonzero boxes in a. This actually denotes the number of active S-boxes in the step Gamma before a. The box weight of a trail ($a \rightarrow A \mid b \rightarrow B \mid c \rightarrow C$) is $\#a + \#b + \#c$.

The box weight of a two-round trail ($a \to A \mid b \to B$) is given by $\#a + \#b$ or equivalently $\#a + \#A$. In other words, the number of active S-boxes in a two round trail is completely determined by the first *step*.

Thanks to the rotational symmetry in the round transformation, it can be seen that if there is a trail ($a \to A \mid b \to B\ldots$) there are in fact 32 such trails, one for each cyclically shifted version of ($a \to A \mid b \to B\ldots$).

Moreover, thanks to the fact that both the linear part and the non-linear part of the round transformation are involutions, for every trail ($a \to A \mid b \to B\ldots$), there exists a corresponding trail of the form: ($B \to b \mid A \to a\ldots$).

9.2 Propagation Analysis

The purpose of the propagation analysis is to provide a lower bound for the correlation coefficient of multiple-round linear trails and the prop ratio of multiple-round differential trails. More precisely, we demonstrate that in any 4-round linear trail, the correlation coefficient is at most 2^{-24} and in any 4-round differential trails the prop ratio is at most 2^{-48}..

In order to demonstrate the mentioned lower bounds, we have searched the complete space of 4-round trails (both linear and differential) with less than 24 active S-boxes. As the maximum input-output correlation of the Gamma S-box is ½, linear trails with 24 active S-boxes or more have a correlation coefficient that is at most 2^{-24}. Likewise, as the maximum input-output probability of the Gamma S-box is ¼, differential trails with 24 active S-boxes or more have a prop ratio that is at most 2^{-48}.

For all 4-round linear traces with less than 24 active S-boxes, we have verified that the correlation coefficient is at most 2^{-24} and for all 4-round differential traces with less than 24 active S-boxes, we have verified that the prop ratio is at most 2^{-48}.

The generation of all 4-round trails (both linear and differential) with less than 24 active S-boxes consists of two phases:

- **Phase 1:** we make an inventory of all steps leading to two-round trails with a small number of active S-boxes.
- **Phase 2:** we chain the two-round trails found in phase 1 to form four-round trails.

9.2.1 Phase 1: two-round trails

The number of active S-boxes in any two-round trail starting with $a \to A$ as its first step is given by $\#a + \#A$.

Basically, we create a database containing all *steps* ($a \to A$) that satisfy at least one of the following conditions (see Figure 1 for a graphic representation):

- Area α: $\#a + \#A < 18$
- Area β: $\#a + \#A < 20$ and $\#a < 7$
- Area β': $\#a + \#A < 20$ and $\#A < 7$

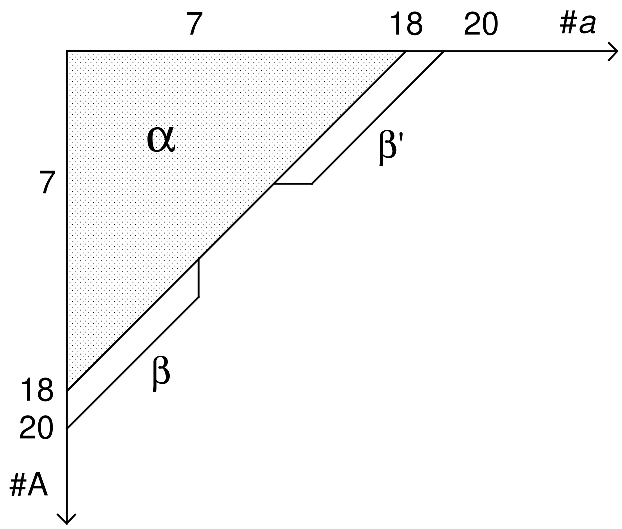

Figure 1 Range of steps in the database

The relation A = Lambda(a) is independent of Gamma and therefore this generation can be done independently of the specific S-box. The generation of this database is a non-trivial exercise. In fact, we make extensively use of the algebraic and symmetry properties of the component transformations, especially Theta to compute it in a reasonable time frame (about one hour on a modern PC).

Table 8 lists the number of steps $a \rightarrow A$, up to rotation equivalence, for the values of #a and #A. As for any step ($a \rightarrow A$) there exists a step ($A \rightarrow a$), the table is symmetric and it does not matter whether #a is on the vertical axis or the horizontal one. We give only the first eight columns of the full table as all other values follow from the symmetry property.

From the table it follows that the only two-round trails with less than 8 active S-boxes are the two trails with 4 active S-boxes (2, 2) and the six trails with 6 active S-boxes (3, 3). Moreover, the table shows the symmetry between ($a \rightarrow A$) and ($A \rightarrow a$).

	1	2	3	4	5	6	7	8
1							4	
2		2				14	4	8
3			6		28	12	70	108
4				163	32	178	328	1,493
5			28	32	617	1,283	3,762	6,261
6		14	12	179	1,283	9,101	15,341	54,660
7	4	4	70	328	3,762	15,341	93,668	273,344
8		8	108	1,493	6,261	54,660	273,344	1,249,658
9		1	357	1,972	21,036	129,640	838,646	4,378,578
10		41	305	5,038	44,593	353,545	2,380,721	**?**
11	1	52	899	9,356	97,629	853,003	**?**	**?**
12		113	1,273	18,489	205,194	2,085,751	**?**	?
13	5	66	1,947	33,605	444,745	4,827,996	?	?
14		149	3,338	63,611	897,923	?	?	?
15		109	5,852	112,168	?	?	?	?
16		199	8,222	?	?	?	?	?
17	2	223	?	?	?	?	?	?
18		?	?	?	?	?	?	?

Table 8: Number of steps $a \rightarrow A$ given $\#a$ and $\#A$.

9.2.2 Phase 2: Chaining two-round trails to four-round trails

Our goal is to find all 4-round trails with less than 24 active S-boxes, or in other words, all trails

$$(a \rightarrow A \mid b \rightarrow B \mid c \rightarrow C \mid d \rightarrow D)$$

with $\#a + \#b + \#c + \#d = \#a + \#A + \#c + \#C < 24$. In our chaining, we cover the entire space of such trails.

We divided the space of four-round trails in two *areas*: Area I and Area II. These areas are indicated in Figure 2.

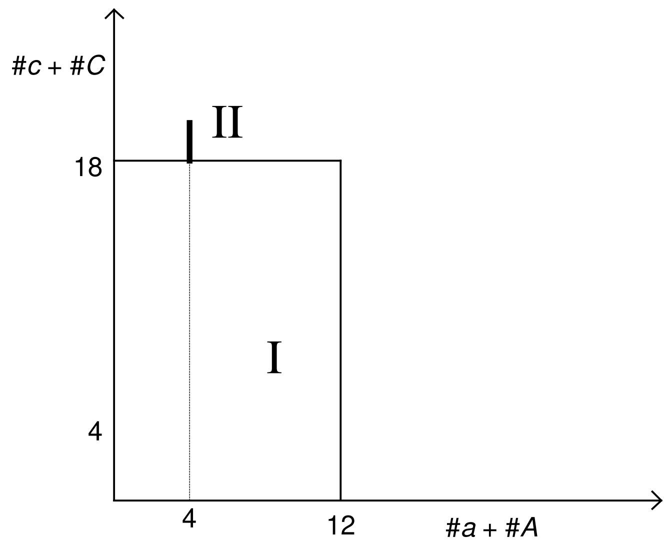

Figure 2 : DC/LC Trails Space

Area I: This covers the four-round trails consisting of a two-round trails with less than 12 active S-boxes, followed by a two-round trail with less than 18 S-boxes.

For each step ($a \rightarrow A$) with $\#a + \#A < 12$ in the database, we do the following:

- We generate all possible patterns b that are compatible with A to give the two-round trail ($a \rightarrow A \mid b \rightarrow B$). The number of steps ($b \rightarrow B$) that are compatible with A is highly dependent on $\#A$. This extension depends on the propagation tables of the Gamma S-box and hence must be done separately for differential and linear trails.
- For each trail ($a \rightarrow A \mid b \rightarrow B$), we check if there is a step ($c \rightarrow C$) in the database that can be chained to it, i.e., for which c is compatible with B. For each step in the database, all 32 rotated versions have to be checked. The check for compatibility consists of two steps:
    - Same position of active S-boxes: B and c must have the same number of active S-boxes and these active S-boxes have to be in the same positions. This already eliminates the vast majority of candidates ($c \rightarrow C$).
    - Compatibility of active S-boxes: each of active S-boxes in the propagation patterns must be compatible. This involves the propagation tables of the Gamma S-box.

Area II: This covers four-round trails consisting of a two-round trail with 4 active S-boxes, followed by a two-round trail with 18 or 19 active S-boxes.

For the 2 trails ($a \to A$) with $\#a + \#A = 4$, we do the following:

- We generate all possible steps $b \to B$ that are compatible with $a \to A$ to give the trail ($a \to A \mid b \to B$).
- If either $\#B < 7$ or $\#B > 12$ (i.e., if we can find ($c \to C$) in the generated database, area Area β or β'), we check if there is a step ($c \to C$) in the database that can be chained to the trail ($a \to A \mid b \to B$).
- If $7 < \#B < 12$, we exhaustively generate all possible ($c \to C$) steps that are compatible with ($a \to A \mid b \to B$) to see whether $\#a + \#A + \#c + \#C < 24$.

Thanks to the symmetry of the of the cipher, for each trail ($a \to A \mid b \to B \mid c \to C$) there exists a trail ($C \to a \mid B \to b \mid A \to a$). Hence by covering area I ($\#a + \#A < 12$, $\#c + \#C < 16$), we cover also another area defined by ($\#c + \#C < 12$, $\#a + \#A < 16$), called area I'. The same can be applied to area II, giving area II'. Figure 3 illustrates the coverage of the space of 4-round trails (below the diagonal) with less than 24 active bytes by the areas I, II, I' and II'.

We can distinguish 4 cases (and we use $0 \le \#a + \#A + \#c + \#C < 24$):

- $0 \le \#a + \#A < 6$: the only possibility is $\#a + \#A = 4$ (see Table 8), implying $0 \le \#c + \#C < 20$. The range [0,18) is covered by area I, the range [18,20) by area II.
- $6 \le \#a + \#A < 12$: this implies $\#c + \#C < 18$. This is completely covered by area I.
- $12 \le \#a + \#A < 18$: this implies $\#c + \#C < 12$. This is completely covered by area I'
- $18 \le \#a + \#A < 24$: this implies $\#c + \#C < 6$. The only possibility is $\#c + \#C = 4$ (see Table 8), implying $18 \le \#a + \#A < 20$. This is completely covered by area II'.

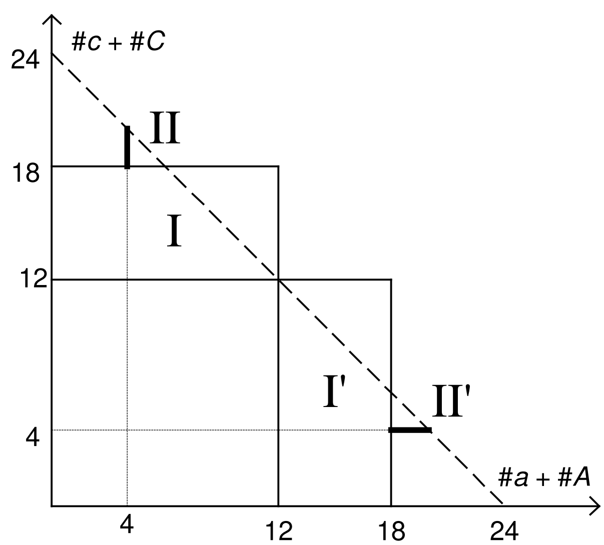

Figure 3 Coverage of the complete space of 4-round trails with 24 active S-boxes