A New Class of Cryptosystems Based on Interconnection Networks

Michael Portz
RWTH Aachen
Lehrstuhl für angewandte Mathematik insb. Informatik
Ahornstr. 55
D-5100 Aachen
michaelp@terpsichore.informatik.rwth-aachen.de

1 Introduction

Interconnection networks have been widely studied in the field of parallel processing. They have one property that makes them very interesting for cryptology, i.e. they give the opportunity to specify and perform permutations at the same time. This paper introduces a new class of cryptosystems which is constructed using boolean functions and interconnection networks. The construction is secure in the sense that it can be used to construct pseudo-random permutation generators from pseudo-random Boolean function generators ([$LuRa$86], [GGM86], [$Schn$88], [$Piep$90], [$Piep$91]). It is proposed, to use simpler functions instead of pseudo-random functions to construct cryptosystems, e.g. theoretical pseudo-random number generators (as proposed by [Yao81],[BBS],etc), practical pseudo-random number generators (linear shift register etc.) or oneway functions [Yao81]. The security of a specific cryptosystem based on boolean functions fulfilling the strict avalanche criterion [$Lloy$89] is investigated.

Chapter 2 gives a short introduction to the theory of interconnection networks. In chapter 3 a new construction of pseudo-random permutation generators from pseudo-random function generators is described. In chapter 4 a new one-key cryptosystem based on interconnection networks and boolean functions fulfilling the strict avalanche criterion [$Lloy$89] is presented. Some arguments for its security are given. An outlook on further research is given in chapter 5. The rest of this introduction is used to give a short review of

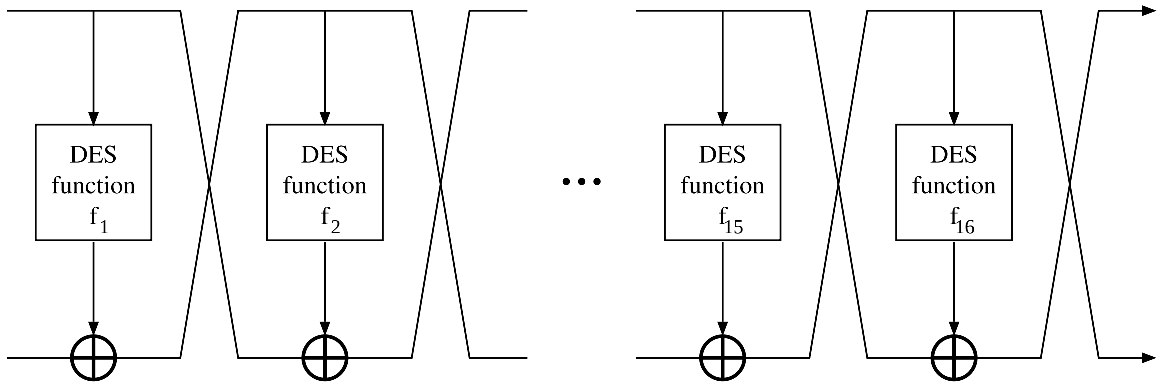


Fig. 0.1: Iterationstructure of DES

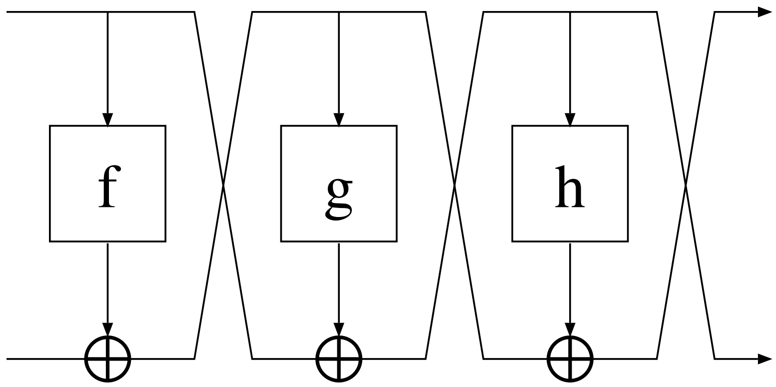


Fig. 0.2: Lubys and Rackoffs Construction

known results. 1.1 contains the results concerning pseudo-randomness and 1.2 the results concerning boolean functions as mentioned above.

1.1 Pseudo-randomness and Permutation Generators

During the last years a lot of work has been spend on the construction of permutation generators. One reason for this is, that most cryptosystems (e.g. RSA, DES) are nothing more than generators of permutations. Specifying a certain key in such a cryptosystem means specifying (or **generating**) a certain permutation on a very large ordered set. On the other hand there has been a cryptographical interest in pseudo-randomness for a long time, too, dealing mostly with the (pseudo)randomness of numbers or bitstrings. In [GGM86] the notion of pseudo-randomness is extended to functions and a construction of pseudo-random functions from pseudo-random bitstrings is described. Luby and Rackoff were the first to look at the special case of pseudo-random **bijective** functions [$LuRa$86][$LuRa$88]. They proved, that such pseudo-random permutations (which bijective functions can be identified with) can be constructed from pseudo-random functions.

The construction given by Luby and Rackoff has the same iterative structure as the DES (Fig.1). Three (instead of 16) of these iterations are performed and in each iteration a different pseudo-random function is used (Fig. 2). Naturally the question arises, whether

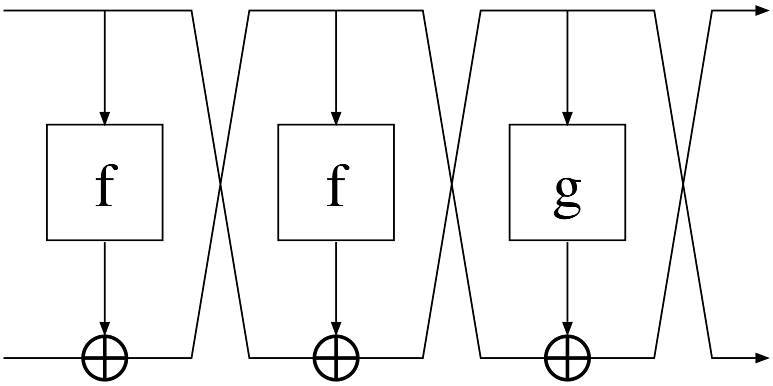


Fig. 0.3: Ohnishis Construction

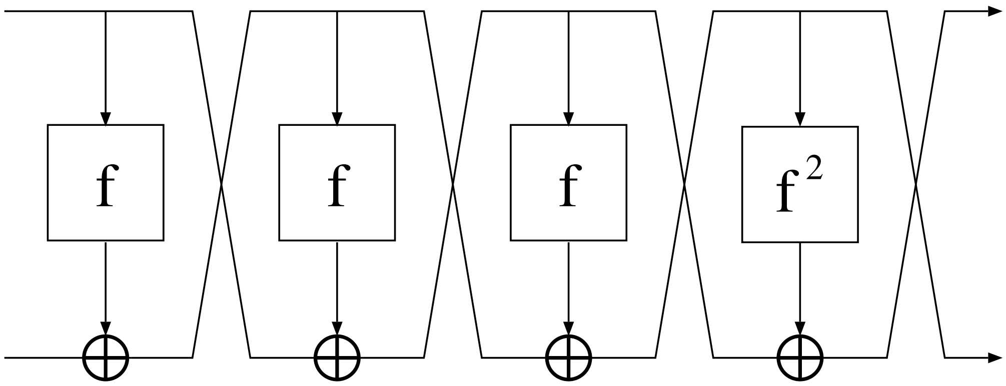


Fig. 0.4: Pieprzyks Construction

the number of functions used can be reduced [$Schn88$]. Zheng, Matsumoto and Imai [$ZMI89$] give a positive answer to that question and proved that two functions are the minimum if three iterations are performed (Fig. 3). Pieprzyk [$Piep90$] was able to perform a construction based on one pseudo-random function and four iterations (Fig. 4).

In this paper only a short review of some of the basic definitions is given. Let throughout the definitions n and k represent any positive integers. With $I = \{0, 1\}$ let F_n and $\mathcal{F}$ be defined as:

$$F_n = \{f \mid f : I^n \rightarrow I^n\} \qquad \mathcal{F} = \bigcup_{n>0} F_n$$

Definition: A **function generator** with key length function $klf(n)$ (klf a polynomial) is a collection $\mathcal{H} = \bigcup_{n>0} H_n$, $H_n = \{h_{n,k} \mid \log k \leq klf(n)\}$, where each $h_{n,k}$ is a function from F_n. It is required that, given a key k of length $klf(n)$, and an input α of length n, $h_{n,k}(\alpha)$ can be computed in time polynomial in n.

The following definition defines function generators mapping to I instead of mapping to I^n. B_n and $\mathcal{B}$ be defined as:

$$B_n = \{f \mid f : I^n \rightarrow I^n\} \qquad \mathcal{B} = \bigcup_{n>0} B_n$$

Definition: A **Boolean function generator** with key length function $klf(n)$ (klf a polynomial) is a collection $\mathcal{H} = \bigcup_{n>0} H_n$, $H_n = \{h_{n,k} \mid \log k \leq klf(n)\}$, where each $h_{n,k}$ is a function from B_n. It is required that, given a key k of length $klf(n)$, and an input α of length n, $h_{n,k}(\alpha)$ can be computed in time polynomial in n.

Definition: A **permutation generator** $\mathcal{H}$ is a function generator such that each function $h_{n,k}$ is bijective. Let $\overline{\mathcal{H}} = \bigcup_{n>0} \overline{H_n}$ where $\overline{H_n} = \{h_{n,k}^{-1}\}$. We say $\mathcal{H}$ is invertible if $\overline{\mathcal{H}}$ is also a permutation generator. In this case, $\overline{\mathcal{H}}$ is the inverse permutation generator for $\mathcal{H}$.

Luby and Rackoff give a formal definition of pseudo-randomness which is based on a definition of "undistinguishability". Informally a function generator is called pseudo-random if for large n no function $h_{n,k}$ can for randomly generated k be distinguished from a really random function.

Definition: A (Boolean) function generator $\mathcal{H}$ is **pseudo-random** if there is no distinguishing circuit family for $\mathcal{H}$. A distinguishing circuit family for $\mathcal{H}$ is an oracle circuit family $\{C_{n_1}, C_{n_2}, \ldots\}$ where $n_1 < n_2 < \ldots$, such that for some polynomial Q_1 and for each n for which there is a circuit C_n the $\{H_n, F_n\}$ (or $\{H_n, B_n\}$ respectively) distinguishing probability of C_n is at least $\frac{1}{Q_1(n)}$.

An oracle circuit family is an infinite family of circuits $\{C_{n_1}, C_{n_2}, \ldots\}$ such that for some polynomial Q_2 and for each n for which there is a circuit C_n:

1. C_n is a circuit which contains the usual boolean gates together with oracle gates, where each oracle gate has n inputs and n outputs. C_n has one boolean output, where the value of the output depends on the way the oracle gates are evaluated. Such a circuit is called an oracle circuit.
2. The size of C_n is less than or equal to $Q_2(n)$ (size defined as number of inter-gate connections)

The $\{H_n, F_n\}$ ($\{H_n, B_n\}$ respectively) distinguishing probability of C_n is defined as follows. Let x_n be the output value of C_n. Let $r_n = P[x_n = 1]$ when a function is randomly chosen from F_n and this function is used to evaluate the oracle gates in C_n. Let $p_n = P[x_n = 1]$ when a key k of length $klf(n)$ is randomly chosen and $h_{n,k}$ is used to evaluate the oracle gates in C_n. Then $d_n = |p_n - r_n|$ is the $\{H_n, F_n\}$ ($\{H_n, B_n\}$ respectively) distinguishing probability of C_n.

Note, that pseudo-random Boolean functions are not explicitely defined in the previous papers, but their existence is implicitly proven in [$GGM86$].

Theorem:*(Luby,Rackoff)* There is a way to construct an invertible pseudo-random permutation generator from a pseudo-random function generator.

In this paper the following new theorem is proven:

New Theorem: There is a way to construct an invertible pseudo-random permutation generator from a pseudo-random Boolean function generator.

1.2 Lloyds result

In her Eurocrypt'89 paper Lloyd gives a formula for estimating the number of boolean functions which fulfil the strict avalanche criterion of order $(m-2)$. Here only boolean function $f : I^m \to I$ are considered. The criterion guarantees, roughly speaking, the cryptographically important property, that changing one input-bit of f or of any subfunction of f leads to a probability 1/2 that the output bit changes, too. The formula she uses to count the functions can as well be used to choose one of these functions randomly. The formula for an n-bit function f (given here for $0-1$-values, Lloyd gives it for the function $\hat{f}(x) := (-1)^{f(x)}$) is as follows:

(Let h be the Hamming weight function (number of 1's in the bitstring representation of its argument), e_i be the i-th unity vector, e_0 be the zero-vector and $x \in I^m$.)

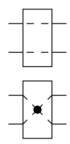
$$f(x) = \frac{h(x)(h(x)-1)}{2} + f(e_0)(h(x)+1) + \sum_{e_i \oplus x > 0; i \in [1:m]} f(e_i)$$

Obviously the function f is completely defined by its values on $\{e_0, \ldots, e_m\}$ and so it is very easy definable and accessible. On the other hand f can be completely reconstructed from knowing only $m+1$ of its function values. This can be done by solving a system of linear equations, taking the values out of $\{e_0, e_1, \ldots, e_m\}$ as variables.

2 Interconnection Networks

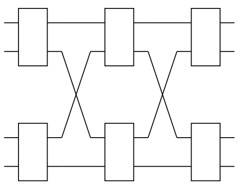
Interconnection networks are commonly dealt with in the context of concurrent processing. They are used to connect a set of input elements (usually processors or memory locations) with a set of output elements (usually processors or memory locations, too). Research on interconnection networks deals in most cases with finding routing algorithms (i.e. how can a given set of connections be realized by a given interconnection network) or finding new topologies which are both efficient and capable of realizing given sets of connections. In our context a different property of interconnection networks is used: they are a very easy way to specify and perform permutations at the same time. The following terminology is according to $[Feng81]$.

2.1 Control strategy

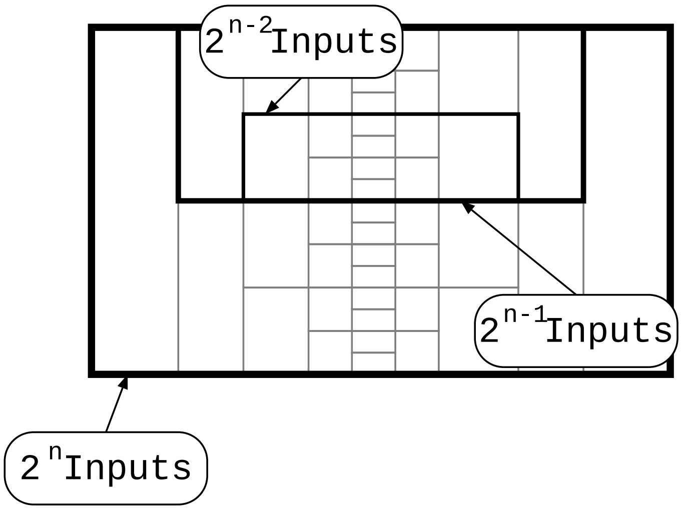


An interconnection network is build up from β- or **switching elements**. A switching element takes two inputs and produces two outputs. Whether the inputs should be exchanged in the specific switching element or not is determined by a control-setting function h. This function maps the unique index of any switching element of the interconnection network onto the set $I = \{0, 1\}$ ($1 \hat{=}$ exchange,$0 \hat{=}$ don't exchange).

2.2 Topology



The topology of an interconnection network specifies how the switching elements are connected to each other. Any topology describes a set of permutations. Choosing the control-setting function h means choosing a specific permutation from this set. This is an interesting property which interconnection networks and cryptosystems have in common. Even more interesting in this context are topologies which are capable of generating all permutations on the input-set. Such networks were introduced by Benes [$Bene65$]. Waksman gives a topology which uses asymptotically the minimal number of switching elements [$Waks68$]. Here the Benes-topology is used, because of its slightly simpler design. At the righthand side a Benes-network for 4 input-elements is shown. Figure 5 shows one for 8 elements. The recursive structure of these networks can easily be seen, so its not stated formally: a Benes-network N_n with 2^n input-elements is build from two Benes-networks N_{n-1} and 2^n additional switching elements.



Continuing this process (N_n contains 2^2 versions of N_{n-2}, N_n contains 2^3 versions of N_{n-3}, etc.) helps introducing the notion of the **internal box of size** 2^k which turns out to be very helpful in section 3. Such an internal box of size 2^k denotes any of the 2^{n-k} Benes-networks N_k which N_n contains. Note that each of these boxes has 2^k inputs, so in this context an internal box is characterized by the number of its inputs. This subsection is completed by an example. Let $h : I^m \rightarrow I$ be a boolean function. Note that in the case of an Benes-network N_n m must at least have the value $\lceil \log(2^{n-1}(2n-1)) \rceil$. In the following example $n = 3$ and $m = 5$. Then $N(h)$ denotes the permutation which results from using h as the control-setting function in a Benes-network of appropriate size. Let further $N(\mathcal{H})$ denote the resulting set of permutations, if $\mathcal{H}$ is a set of boolean functions. Figure 5 shows an example of an interconnection network with Benes-topology and control-setting function $h : I^5 \rightarrow I$ and resulting permutation $N(h)$.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...	31
h(x)	1	0	0	0	1	1	0	0	0	1	1	1	0	0	1	0	1	0	0	1	x	...	x

$$N(h) = (0, 5, 3, 6, 4, 2, 1, 7)$$

2.3 Virtual Interconnection Networks

To be ready to investigate the use of interconnection networks in the design of cryptosystems one has to overcome one last difficulty: If one chooses the input size of an interconnection network according to the security constraints normally put on cryptosystems, one cannot establish the interconnection network physically any more. Cryptosystems like DES and RSA usually describe sets of permutations on 2^{64} elements or $\sim 2^{512}$ elements respectively. A solution to this is dealing with a virtual interconnection network, which could be derived from the topology.

A virtual interconnection network only realizes those parts of an interconnection network, which are absolutely necessary to compute a certain output. This is done by additionally defining the next-index-function ni. This function computes the index of the next switching

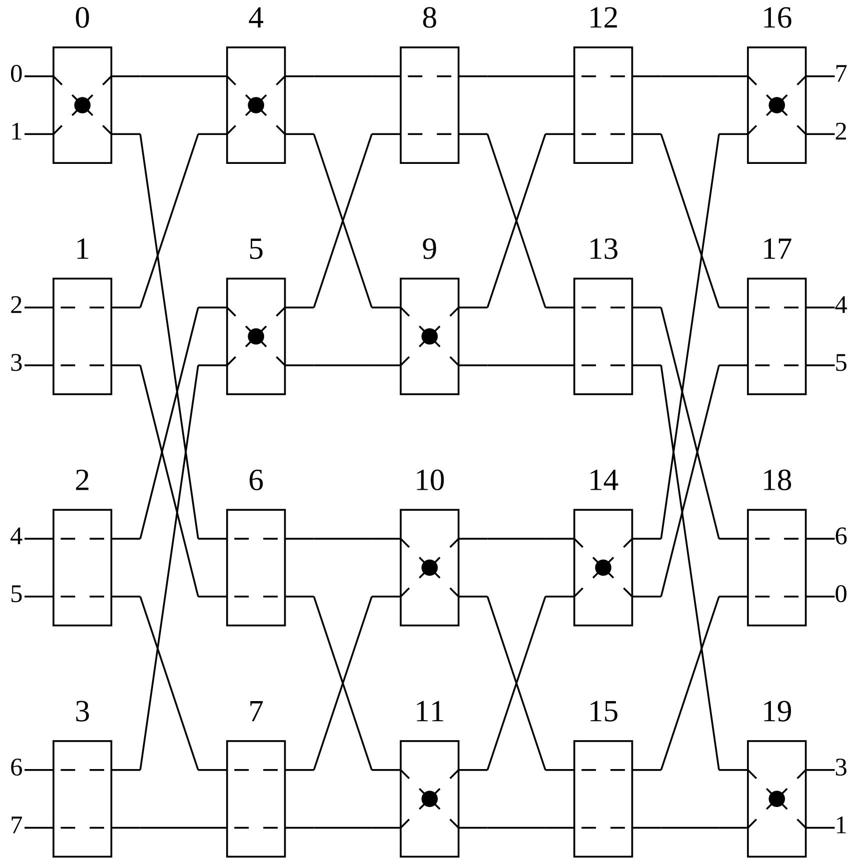


Fig. 0.5: Activated Benes-Network with 8 Inputs

element out of the index of the present switching element and its relative output position (*upper* or *lower*). Figure 6 shows how a virtual interconnection network for N_3, h and the inputs 0 and 3 is established. The next-index-function ni has e.g. the values $n(6, lower) =$ 11 and $n(6, upper) = 10$.

Finally a remark concerning the complexity of a N_n-generated permutation ($n \in \mathbb{N}$) has to be made : The computational complexity K of a permutation $N_n(h_n)$ depends on the complexity of the control-setting function h_n and the next-index-function ni_n in the following way:

$$K(N_n^{h_n, ni_n}) = (2n - 1) \cdot (K(h_n) + K(ni_n))$$

3 The New Construction

The new construction can be described very short: Let $\mathcal{H}$ be a pseudo-random Boolean function generator. Then $N(\mathcal{H})$ should be the constructed permutation generator.

Theorem: If $\mathcal{H}$ is a generator for pseudo-random Boolean functions, then $N(\mathcal{H})$ is an invertible pseudo-random permutation generator.

This theorem is proven as corollary to the following lemma:

Lemma 1: There is no distinguishing circuit family for $N(\mathcal{B})$ and $\mathcal{S}$.

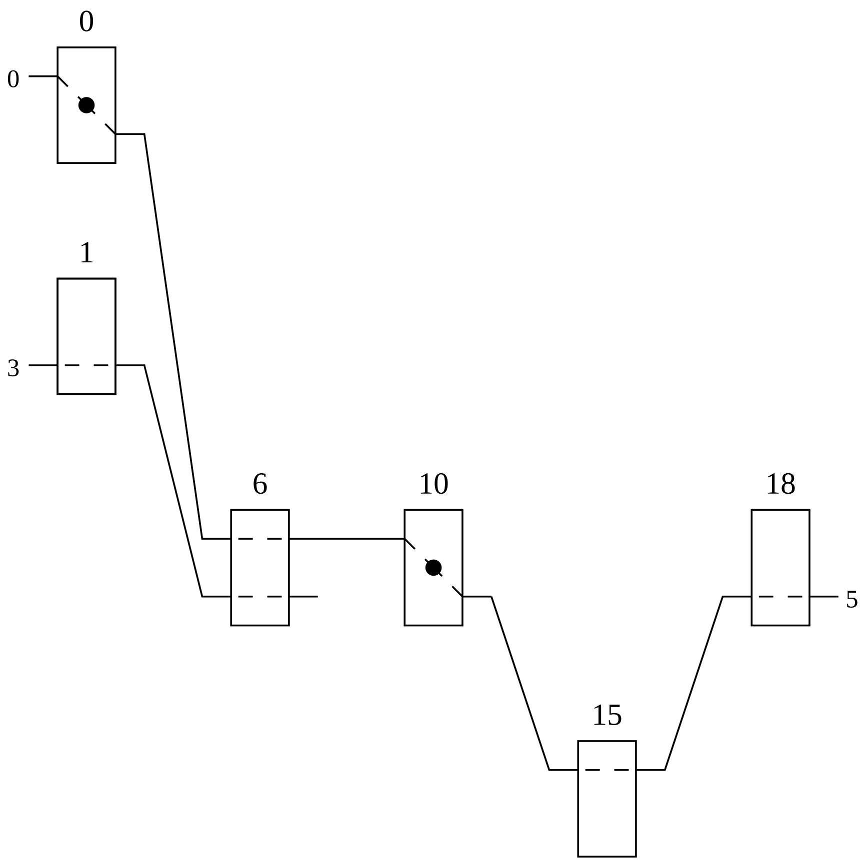


Fig. 0.6: Virtual Version of Fig 5 for inputs 0 and 3.

The proof of the theorem shows that if lemma 1 holds and if there were a distinguishing circuit family for $N(\mathcal{H})$ and $\mathcal{S} = \bigcup_{n>0} S_{2^n}$ there would be a distinguishing circuit family for $\mathcal{H}$ and $\mathcal{B}$ as well, contradicting the assumption that there is none.

3.1 Proof of the Lemma 1

The idea of the proof is to prove that the output-distribution of any polynomial-sized sample of $N(f)$ is approximately the same as the output-distribution of any polynomial-sized sample of $\pi \in S_{2^n}$, namely the Laplace-distribution. The following remarks give a first impression of the proof technique.

Remark 1: Each single input to a Benes-network is uniformly distributed onto the outputs by $N(f)$, if f is a uniformly distributed random variable mapping to B_n for some n.

Remark 2: The decision, whether an input is mapped to the upper half or to the lower half of the output-set $[0 : 2^n - 1]$ is made in the middle (the n-th) column (Fig.7). A similiar statement holds for all further "halfs": the decision, whether an input is mapped to the upper half or the lower half of the half which has been chosen in column n is made in column $n+1$. The decision, whether an input is mapped to the upper half or the lower half of the quarter (half of half) which was specified in columns n and $n+1$ is made in column $n+2$, and so on. From this remark it follows immediately, that if two values decide

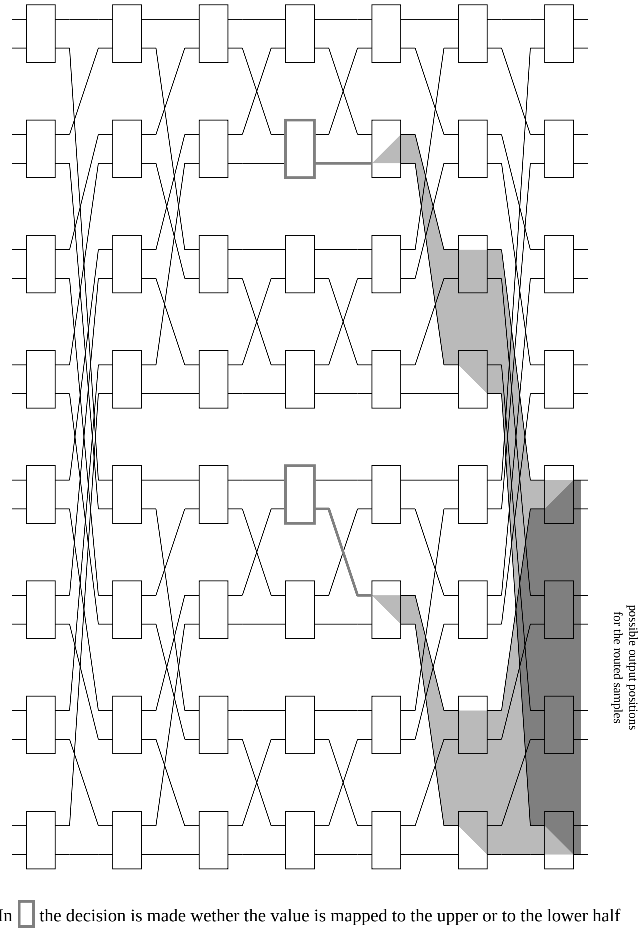


Fig. 0.7: Illustration of remark 2, N_4

for different output-intervals, they cannot collide anywhere behind the column, where they made their decision.

One special case, which is of interest for the rest of the proof, is the situation in column $\frac{n}{2}+n-1$: in this column the decision is made, into which intervall of size $2^{\frac{n}{2}}$ each output is mapped. Note, that this column is the last column of the internal boxes of size $2^{\frac{n}{2}}$ (see section 2.2).

Remark 3: No matter what $k \leq 2^{\frac{n}{2}}$ sample points one chooses, the probability that not two of them will enter the same internal box in column $2^{\frac{n}{2}}$ is bigger than $\frac{(2^{\frac{n}{2}})^{\underline{k}}}{(2^{\frac{n}{2}})^k}$.

From these three remarks it follows, that for large n (virtually with probability 1):

- each value chooses a different internal box of size $2^{\frac{n}{2}}$ (remark 3).
- thus the values are mapped uniformly distributed onto the outputs of these internal boxes (remark 1).
- they are mapped uniformly distributed onto the output-intervals of size $2^{\frac{n}{2}}$ (remark 2).
- there will be no collision in the columns starting from $\frac{n}{2}$ up to $2n-1$, which is the last one (remark 3 and remark 2).

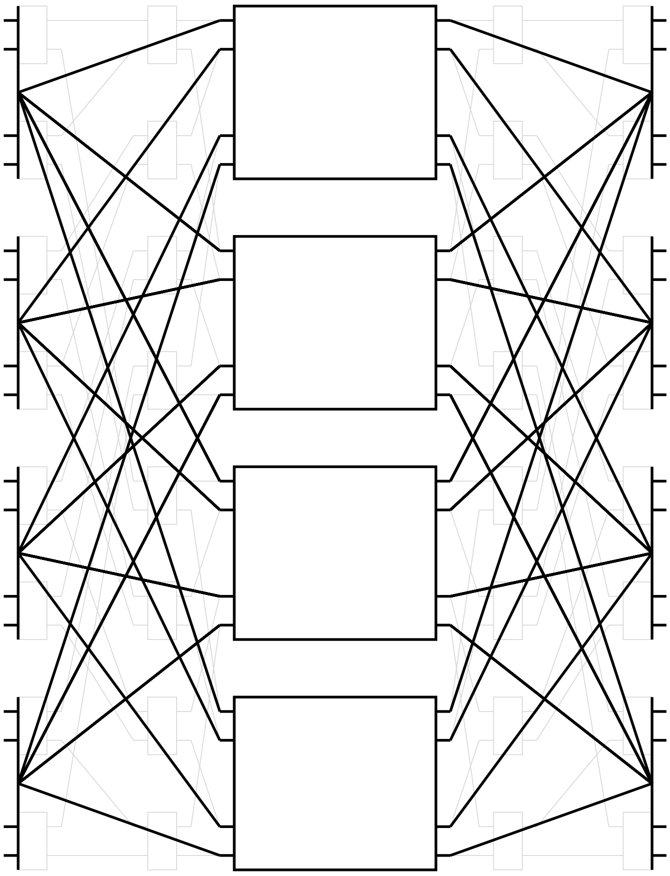


Fig. 0.8: Correspondence of internal boxes and output intervals

These remarks and their conclusions roughly describe the proof. They show that it is simply not possible to choose inputs, so that one can learn anything on the resulting outputs of the permutation. The reason for that is, that as long as one is restricted to polynomially many (in n) inputs, the input-values are routed independently through the main part of the network.

Preliminaries: Let throughout the rest of this section denote $B_m = \{f : I^m \to I\}$ and $\mathcal{B} = \bigcup_m B_m$. To emphasize the fact, that m depends on n as described in subsection 2.3, m is replaced by $l(n)$ as follows: For real numbers r let from now on I^r denote the set $[0 : \lfloor 2^r \rfloor - 1]$. If $r \in N$ nothing changes; if $r = \log x \notin N$ although one cannot deal anymore with bitstrings, one has at least a set which contains the right number of elements. Now we are able to replace m by $l(n) = \log(2^{n-1}(2n-1))$.

First of all one has to define an appropriate probability space. Here the space must describe the following event. For a certain n

- with probability $\frac{1}{2}$ one set out of $\{B_{l(n)}, S_{2^n}\}$ is chosen.
- from the chosen set one element is drawn uniformly distributed.

To describe this experiment it turns out to be best to choose

$$\begin{aligned}
\Omega_{n,1} &= B_{l(n)} \\
\Omega_{n,2} &= S_{2^n} \\
P_{n,i} &= \mathcal{L}(\Omega_{n,i}) \quad , i \in \{1,2\} \\
\Omega_n &= \Omega_{n,1} \cup \Omega_{n,2} \\
P_n(A) &= \frac{P_{n,1}(A \cap \Omega_{n,1}) + P_{n,2}(A \cap \Omega_{n,2})}{2} \quad , A \subset \Omega_n
\end{aligned}$$

with $\mathcal{L}$ being the Laplace-distribution. Further some notations have to be given, to be able to describe certain events in this probability space.

Notations: The probability, that a random function mapping to a set of 2^l different elements does not map any two of k randomly picked arguments onto the same value, is described by:

$$G_{l,k} = \frac{(2^l)^{\underline{k}}}{(2^l)^k}$$

where $(x)^{\underline{k}}$ denotes the Pochhammer-symbol which is defined as:

$$(x)^{\underline{k}} = x \cdot (x-1) \cdot \ldots \cdot (x-k+1)$$

The notation π_ω describes the permutation which results from ω. Formally:

$$\pi_\omega = \begin{cases} N(\omega) & \omega \in \Omega_{n,1} (= B_{l(n)}) \\ \omega & \omega \in \Omega_{n,2} (= S_{2^n}) \end{cases}$$

Additionally, if ω is a control-setting function (i.e. drawn from $\Omega_{n,1}$), π_{ω_l} denotes the permutation one gets, if one cuts the Benes-network vertically at the inputs of the $\frac{n}{2}+1$-th column. Thus ω_l is the subfunction of the control-setting function ω which suffices to compute the settings of the switching elements in the first $\frac{n}{2}$ columns.

In section 2.2 the notion of an "internal box of size 2^k" was introduced. In the same way the input set and the output set are divided into "intervals of size 2^k". As the following lemmata and definitions only deal with internal boxes, input intervals and output intervals of size $2^{\frac{n}{2}}$, the addendum "of size $2^{\frac{n}{2}}$" is left out.

Definition: The (partial) output $\begin{bmatrix} o_1 \\ \vdots \\ o_k \end{bmatrix}$ of a permutation is said to fulfil the **least distance condition** (LDC), if not two of the $\{o_1, \ldots, o_k\}$ are in the same output interval (of size $2^{\frac{n}{2}}$).

The following random variables and subsets of Ω_n are defined:

$$\begin{aligned}
X_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}}(\omega) &= \begin{bmatrix} o_1 \\ \vdots \\ o_k \end{bmatrix} \quad \text{with:} \quad o_j = \pi_\omega(i_j), 0 \le i_1 < i_2 < \ldots < i_k < 2^n \\
L1_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}} &= \{\omega \mid \pi_{\omega_l}(\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}) \quad \text{fulfils the LDC}\} \\
L2_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}} &= \{\omega \mid \pi_{\omega}(\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}) \quad \text{fulfils the LDC}\}
\end{aligned}$$

The set $L1_n^{\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]}$ describes those control-setting functions, which map each input $i_j, j \in [1:k]$ into a private (= different) internal box, whereas $L2_n^{\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]}$ contains those ω, which spread the inputs good enough regarding the output intervals. With these notations and definitions it is now possible to formulate the necessary lemmata.

Lemma 2: Let Q and q be any polynomials. There is no probabilistic algorithm which is restricted to at most $k < q(n)$ queries to an oracle realizing a permutation π_ω, ω chosen according to P_n from Ω_n, which can choose inputs $\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]$ so that for more than finite many n:

1 $$P(\omega \notin L2_n^{\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]} \,|\, \omega \in \Omega_{n,2}) \geq \frac{1}{Q(n)} \quad \text{and further:}$$

2 $$P(\omega \notin L2_n^{\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]} \cap L1_n^{\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]} \,|\, \omega \in \Omega_{n,1}) \geq \frac{1}{Q(n)}$$

Proof:
Proof of 1.: Each input an algorithm chooses is independently randomized by the permutation $\omega \in \Omega_{n,2}$, so there is no "better way" of computing such inputs than trying. The probability of the first input to be mapped to a private output interval is 1. The second input is mapped to an output which is not in the same interval as the first input with probability
$$\frac{(2^{\frac{n}{2}} - 1) \cdot 2^{\frac{n}{2}}}{2^n - 1}$$
Continuing this argumentation the j-th input is mapped to a different interval than the first $j-1$ inputs with probability

$$\frac{(2^{\frac{n}{2}} - j + 1) \cdot 2^{\frac{n}{2}}}{2^n - j + 1}$$

Multiplying these probabilities results in a probability for the mapping being injective relative to the output intervals:

$$\frac{2^{\frac{n}{2}} \cdot 2^{\frac{n}{2}}}{2^n} \cdot \frac{(2^{\frac{n}{2}} - 1) \cdot 2^{\frac{n}{2}}}{2^n - 1} \cdot \ldots \cdot \frac{(2^{\frac{n}{2}} - k + 1) \cdot 2^{\frac{n}{2}}}{2^n - k + 1}$$

which can be transformed to
$$\frac{G_{\frac{n}{2},k}}{G_{n,k}}$$

Proof of 2.: In this case each input is mapped to the outputs "nearly" independently from the already chosen ones. If a certain intermediate state of the network occurs, the inputs are mapped to the outputs independently. The probability, that this intermediate state indeed occurs is next to 1.

If $L1_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]}$ already happened one knows, that each input value has its private internal box. As each Benes-network maps a single input uniformly distributed to its outputs, each of the k input values is mapped independently onto the output intervals. Note, that each of the $2^{\frac{n}{2}}$ outputs of a internal box corresponds to a certain output interval (Remark 2, Fig.8). As there are no other restrictions on the ω than that it is in $L1_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]}$ the probability for $(L2_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]} \mid L1_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]})$ to happen is $G_{\frac{n}{2},k}$.

If each input value i_j has its private input interval, each of them is independently mapped to the internal boxes. The probability that two of the input values will enter the same internal box is again $G_{\frac{n}{2},k}$.

If two (or more) input values are chosen from the same input interval not two (or more) of them can enter the same internal box, as only one value from each input interval can enter a certain internal box. Thus the probability of $L1_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]}$ increases, the more input values are chosen from the same interval, reaching its maximum (namely 1), iff alle input values are chosen from the same input interval.

Up to now the following values are known:

$$P(\omega \in L2_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]} \mid \omega \in \Omega_{n,2}) = \frac{G_{\frac{n}{2},k}}{G_{n,k}}$$

$$P(\omega \in L2_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]} \mid \omega \in L1_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]} \cap \Omega_{n,1}) = G_{\frac{n}{2},k}$$

$$P(\omega \in L1_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]} \mid \omega \in \Omega_{n,1}) \geq G_{\frac{n}{2},k} \quad \text{, and thus}$$

$$P(\omega \in L2_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]} \cap L1_n^{\left[\begin{smallmatrix} i_1\\ \vdots\\ i_k\end{smallmatrix}\right]} \mid \omega \in \Omega_{n,1}) \geq G^2_{\frac{n}{2},k}$$

As

$$G_{n,k} \leq 1, \quad G_{\frac{n}{2},k} \leq 1, \quad G_{\frac{n}{2},k} \leq G_{n,k}, \quad \frac{G_{\frac{n}{2},k}}{G_{n,k}} \geq G_{\frac{n}{2},k} \geq G^2_{\frac{n}{2},k}$$

it suffices to proof that for every polynomial Q and large enough n

$$1 - G^2_{\frac{n}{2},k} < \frac{1}{Q(n)}$$

The following three estimations are very helpful for the completion of the proof. Let $A_{\frac{n}{2},k}$ be defined as

$$A_{\frac{n}{2},k} = \frac{(k-1)k}{2 \cdot 2^{\frac{n}{2}}} + \frac{(k-1)k(2k-1)}{6 \cdot 2^n}$$

$$\ln(1-x) \leq -x$$
$$\ln(1-x) \geq -(x+x^2)$$
$$G_{\frac{n}{2},k} \geq e^{-A_{\frac{n}{2},k}}$$

The first two hold at least for $x \in (0, 0.5)$ and can be derived from the Taylor series of ln. The third is proved as follows:

$$\begin{aligned} G_{\frac{n}{2},k} = \frac{(2^{\frac{n}{2}})^{\underline{k}}}{(2^{\frac{n}{2}})^k} &= \prod_{0 \leq l < k} 1 - \frac{l}{2^{\frac{n}{2}}} \\ &= e^{\sum_{0 \leq l < k} \ln(1 - \frac{l}{2^{\frac{n}{2}}})} \\ &\geq e^{-\sum_{0 \leq l < k} (\frac{l}{2^{\frac{n}{2}}} + \frac{l^2}{(2^{\frac{n}{2}})^2})} \\ &= e^{-A_{\frac{n}{2},k}} \end{aligned}$$

Finally:

$$\begin{aligned} (G^2_{\frac{n}{2},k} > 1 - \frac{1}{Q(n)}) &\iff (2 \ln G_{\frac{n}{2},k} > \ln(1 - \frac{1}{Q(n)})) \\ &\Leftarrow (2 \ln G_{\frac{n}{2},k} > -\frac{1}{Q(n)}) \\ &\Leftarrow (2 \ln e^{-A_{\frac{n}{2},k}} > -\frac{1}{Q(n)}) \\ &\iff (2A_{\frac{n}{2},k} < \frac{1}{Q(n)}) \end{aligned}$$

With increasing n and for $k < q(n)$ the value $2A_{\frac{n}{2},k}$ approaches 0 faster than any $\frac{1}{Q(n)}$, for any polynomial q. Thus there can only be finite many n which do not fulfil 1 or 2.

Lemma 3: For all $\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}$ and $\begin{bmatrix} o_1 \\ \vdots \\ o_k \end{bmatrix}$ the following holds:

$$P(X_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}} = \begin{bmatrix} o_1 \\ \vdots \\ o_k \end{bmatrix} \mid L2_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}}, L1_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}}, \Omega_{n,1}) = P(X_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}} = \begin{bmatrix} o_1 \\ \vdots \\ o_k \end{bmatrix} \mid L2_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}}, \Omega_{n,2})$$

Proof: Any $\begin{bmatrix} o_1 \\ \vdots \\ o_k \end{bmatrix}$ which does not fulfil the LDC has probability 0 in both cases. Any other output has the probability

$$\frac{1}{(2^{\frac{n}{2}})^{\underline{k}}(2^{\frac{n}{2}})^k}$$

in both cases because the j-th input $i_j, 1 \leq j \leq k$ has to find the correct output interval (the correct output position of its private internal box respectively, remark 3) out of the $(2^{\frac{n}{2}}) - j + 1$ remaining. If it succeeds it still has to find the right output position out of the $2^{\frac{n}{2}}$ remaining. Note that the preconditions on the left hand side of the equation guarantee, that for each value the mapping onto the output intervals happens independently from the others.

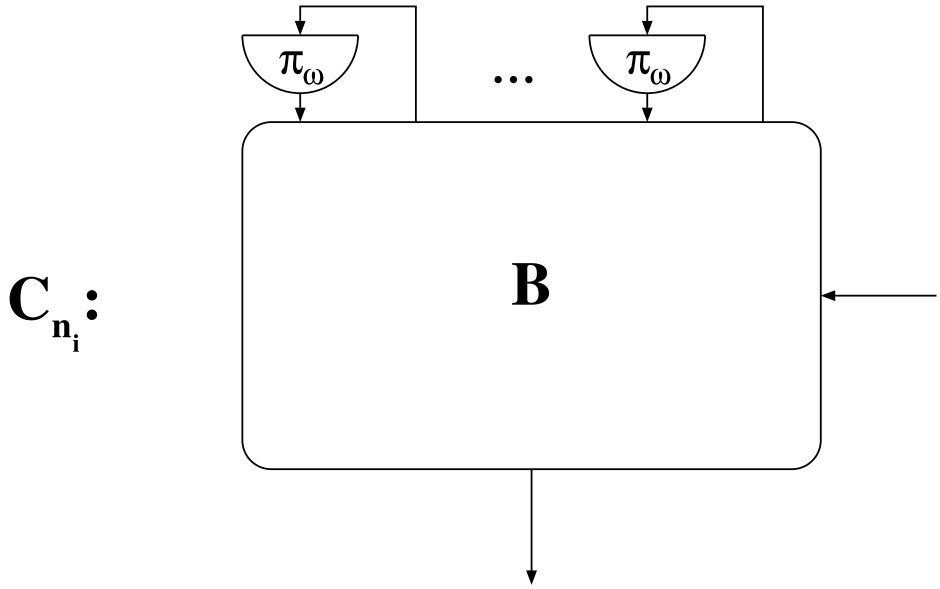


Fig. 0.9: Distinguishing Circuit for C

The following corollary is a more formal version of lemma 1:

Corollary: For each oracle circuit family $\{C_{n_1}, C_{n_2}, \ldots\}$, each circuit C_{n_i} with at most $q(n_i)$ many oracle gates, and for each choice of polynomials Q and q there is an n_0 so that for all $n_i > n_0$ the circuit C_{n_i} has a $\{N(B_{l(n)}), S_{2^n}\}$ distinguishing probability of less than $\frac{1}{Q(n_i)}$.

Proof: According to lemma 2 there are no algorithms which can avoid (with probability bigger than $\frac{1}{Q(n)}$ for any polynomial Q) that $\omega \in (L2_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}}, L1_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}}, \Omega_{n,1})$ or $\omega \in (L2_n^{\begin{bmatrix} i_1 \\ \vdots \\ i_k \end{bmatrix}}, \Omega_{n,2})$ respectively if the algorithms are restricted to at most $k < q(n)$ samples for n bigger than n_0. Especially (as we are not restricted to polynomial time algorithms) there are no boolean circuits with at most $q(n)$ oracle gates which can succeed in doing this. Lemma 3 shows, that if this is the case the distribution of the outputs are the same, no matter whether the permutation is chosen from $\Omega_{n,2} = S_{2^n}$ or from $\Omega_{n,1} = B_{l(n)}$. Thus the corollary holds.

3.2 Proof of the Theorem

The proof of the theorem now can make use of lemma 1. Let again be $\mathcal{H}$ be a pseudo-random Boolean function generator, $\mathcal{B}$ be the set of all Boolean functions and $\mathcal{S}$ be the set of all permutations. It shows, that if there is no distinguishing circuit for $\mathcal{S}$ and $N(\mathcal{B})$ (i.e. lemma 1 holds) and if there were a distinguishing circuit family for $\mathcal{S}$ and $N(\mathcal{H})$ there were a distinguishing circuit family for $\mathcal{H}$ and $\mathcal{B}$ as well. Note that the invertibility is trivially achieved using this construction.

Proof of Theorem: Let $C = \{C_{n_1}, C_{n_2}, \ldots\}$ be a distinguishing circuit family for $N(\mathcal{H})$ and S. Figure 9 shows one of the circuits C_{n_i} with all oracle gates lined up at the top.

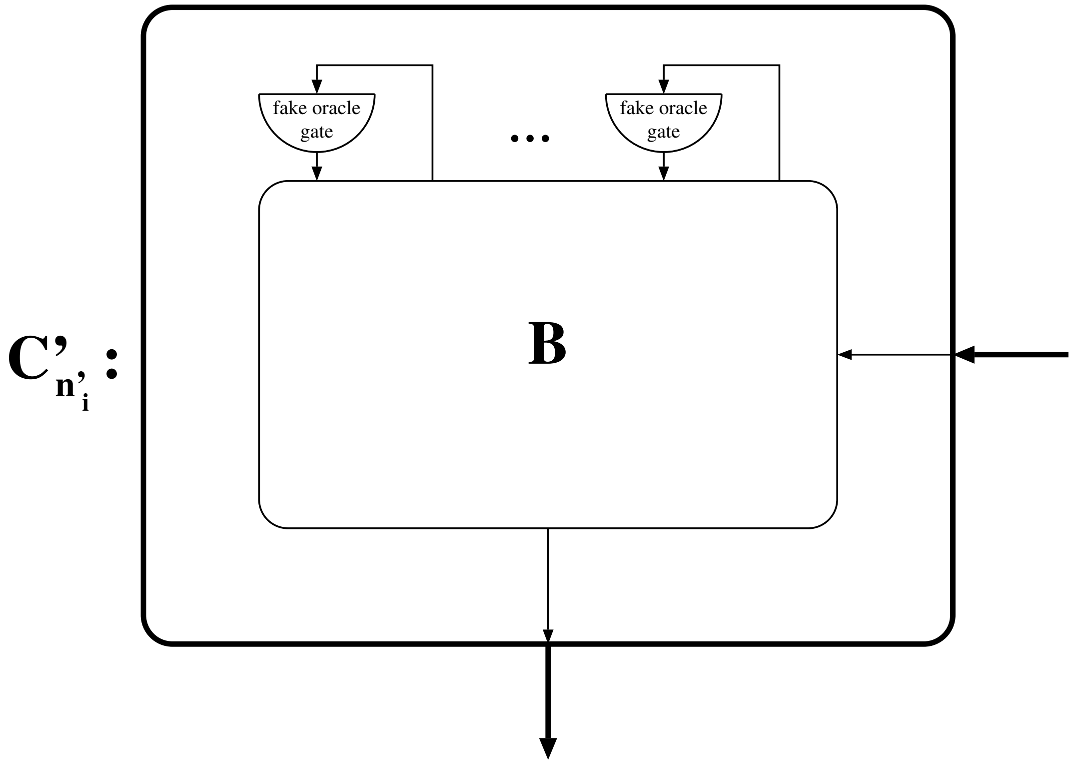


Fig. 0.10: Distinguishing Circuit for C''

As there is no distinguishing circuit family for $N(\mathcal{B})$ and S, any circuit C_{n_i} with $n_i > n_0$ (n_0 as in lemma 2) can distinguish between functions from $N(\mathcal{H})$ and functions from $N(\mathcal{B})$, too (otherwise it would contradict the lemma 1). Thus the oracle circuit family $C' = \{C_{n'_1}, C_{n'_2}, \ldots\}$ with

$$n'_1 = \min\{n_i > n_0\} \quad \text{and}$$
$$n'_{j+1} = n_{i+1} \quad \text{, if} \quad n'_j = n_i$$

is a distinguishing circuit family for $N(\mathcal{H})$ and $N(\mathcal{B})$. From C' a oracle circuit family $C'' = \{C'_{n'_1}, C'_{n'_2}, \ldots\}$ can be constructed using fake oracle gates. Figure 10 shows one of the circuits $C'_{n'_i}$. The fake oracle gates are themselves oracle circuits (Fig. 11) which compute a permutation $N(h)$ if h is used to evaluate the oracle gates. Then C'' is a distinguishing circuit family for $\mathcal{H}$ and $\mathcal{B}$. Note that all used sub-circuits (ni, $i\ Mod$ 2,etc.) are polynomially-sized circuits. This completes the proof.

4 Other Control-setting Functions

In this context naturally one question arises: how simple can the functions be which are used as control-setting functions without making the resulting cryptosystem crytpographically insecure. One proposal which is made here is to use “simple” boolean functions mapping I^m to I which fulfil the strict avalanche criterion of order $m-2$. This properties guarantees a “statistical perfectness”, in that changing one input bit means that the output bit changes exactly with probability $\frac{1}{2}$. S.Lloyd gives an enumeration, which shows,

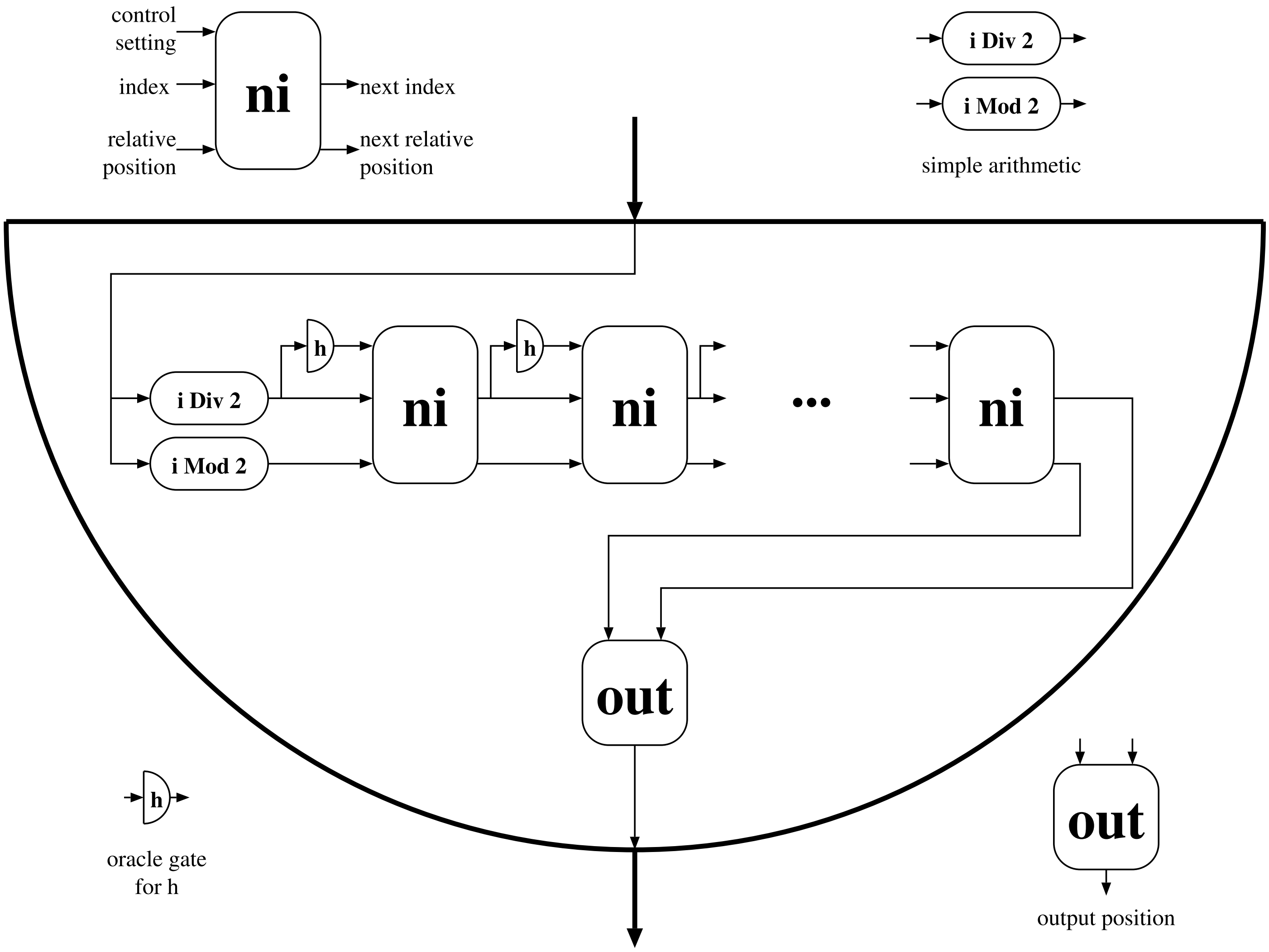


Fig. 0.11: Fake Oracle Gate for $N(h)$

that there are 2^{m+1} such functions [$Lloy$89]. Each of these functions can be expressed as

$$f(x) = \frac{h(x)(h(x)-1)}{2} + f(e_0)(h(x)+1) + \sum_{e_i \oplus x > 0; i \in [1:m]} f(e_i)$$

for an appropriate choice of $f(e_j), j \in [0:m]$, e_0 being the zero-vector and $e_j, j > 0$ being the j-th unity vector in I^m. h denotes the Hamming-weight function, which computes the number of 1's in the binary representation of its argument.

These functions are easy in that each function is easily accessible and each function value is computable fast. Only $m+1$ bits (namely $f(e_j), j \in [0:m]$) are needed to define such a function. All operations which are involved in computing any function value are done modulo 2. The most complex computation is the one of $h(x)$, which is nevertheless fast.

One has to be careful, if one wants to use such simple functions. Only $m+1$ function values $x_0, \dots, x_m$ are needed to reconstruct the whole function. This can be done by solving the following set of linear equations ($j \in \{0, \ldots, m\}$):

$$f(e_0)(h(x_j)+1) + \sum_{e_i \oplus x_j > 0; i \in [1:m]} f(e_i) = f(x_j) - \frac{h(x_j)(h(x_j)-1)}{2}$$

The main argument for the security of such a system is, that interconnection networks prevent the cryptanalyst from completely determining any value $f(x_j)$ of the control-setting function. So he will never be absolutely sure, which set of linear equations he has to solve.

Up to now it is not known, whether there is a less expensive way of choosing the correct set of equations than doing an exhaustive search, assuming a chosen plaintext attack. The system is breakable, if one assumes a combined chosen plaintext/chosen ciphertext attack, but this kind of attack does not seem to be very realistic. Even this attack can be prevented, if one uses additional design criteria on the network [BHP91].

5 Future Research

In section 1 it has already been pointed out, that cryptosystems and interconnection networks are related in that both are able to generate permutations. The main question for future research is, what classes of "cryptograpically interesting" boolean functions define secure cryptosystems, if one uses them as control-setting functions for a Benes-network.

A related question is, whether it is possible to construct pseudo-random permutation generators immediately from pseudo-random number generators or even from oneway-functions? A positive answer to this question would allow more efficient cryptosystems to be built which are "provable secure". The author conjectures that both constructions are possible if one uses interconnection networks.

Klingler [$Klin$91] pointed out, how techniques as described by [$OpTs$71] can possibly simplify the proof of the main theorem. This new proof may help to give a positive answer to the question above. The results will be published in a forthcoming paper.

Finally it must be pointed out, that Benes-networks are only one possible network topology, and that there are numerous other topologies with different properties, which possibly are useful for the construction of cryptosystems, too.

6 Conclusion

The notion of interconnection networks has been viewed from the cryptographic point of view. It has been shown, how to construct a set of permutations out of a set of Boolean functions. The appropriateness of two classes of Boolean functions as control-setting functions has been investigated: Pseudo-random Boolean functions as control-setting functions result in pseudo-random permutations and Boolean functions fulfilling the strict avalanche criterion of higher order result in fast computable permutation generators, which possibly turns out to be a good cryptosystem. Finally it has been proposed to use different network topologies and different Boolean functions to define cryptosytems.

7 References

[BBS] L.Blum, M.Blum, M.Shub: "A simple unpredictable pseudo-random number generator", SIAM Journal on Computing, Vol. 15, 1986, pp. 364-383

[$Bene65$] V.E.Benes: "Mathematical theory of connecting networks and telephone traffic", Academic Press, New York, 1965

[$BHP91$] T.Beth, P.Horster, M.Portz: "Verbindungsnetzwerke in der Kryptologie"; to appear as report of the European Institute for System Security (E.I.S.S.)

[$Feng81$] Tse-yun Feng: "A survey of interconnection networks"; Computer; IE^3; Dec 1981, pp. 12-27

[$GGM86$] O.Goldreich, S.Goldwasser, S.Micali: "How to construct random functions"; Journal of the ACM, Vol. 33, No. 4, Oct. 1986, pp. 792-807

[$Klin91$] T.Klingler, private communication

[$Lloy90$] S.Lloyd: "Counting Functions satisfying a higher order strict avalanche criterion"; Advances in Cryptology - EUROCRYPT '89, Lecture Notes in Computer Science 434, Springer Verlag, 1990

[$LuRa86$] M.Luby, C.Rackoff: "Pseudo-random permutation generators and cryptographic composition"; Proceedings of the 18th ACM Symposium on the Theory of Computing, ACM, 1986, pp. 356-363

[$LuRa88$] M.Luby, C.Rackoff: "How to construct pseudo-random permutations from pseudorandom functions"; SIAM Journal of Computing, Vol. 17(2), 1988, pp. 373-386

[$OpTs71$] D.C.Opferman, N.T.Tsao-Wu: "On a class of rearrangeable switching networks" "Part I: Control Algorithm"; The Bell System Technical Journal, Vol. 50, No. 5, May-June 1971, pp. 1579-1600; "Part II: Enumeration Studies and Fault Diagnosis"; The Bell System Technical Journal, Vol. 50, No. 5, May-June 1971, pp. 1601-1618

[$Ohni88$] Y. Ohnishi: "A study on data security"; Master Thesis (in Japanese); Tohoku University, Japan, 1988

[$Piep90$] J.Pieprzyk: "How to construct pseudorandom permutations from single pseudorandom functions"; Advances in Cryptology - EUROCRYPT '90, Lecture Notes in Computer Science 473, Springer Verlag, 1991

[$Piep91$] J.Pieprzyk: "Pseudorandom authentication systems"; pre-proceedings of EUROCRYPT '91, Brighton, April 1991

[$Port91$] M. Portz: "On the use of interconnection networks in cryptography"; to appear in: Proceedings of EUROCRYPT '91, Brighton, April 1991

[$Schn88$] C.P.Schnorr: "On the construction of random number generators and random function generators"; Advances in Cryptology - EUROCRYPT '88, Lecture Notes in Computer Science 330, Springer Verlag, 1989

[$Waks68$] A.Waksman: "A permutation network"; Journal of the ACM, Vol. 15, No. 1, Jan. 1968, pp. 159-163

[$ZMI89$] Y. Zheng, T. Matsumoto, H. Imai: "On the construction of block ciphers provably secure and not relying on any unproved hypothesis"; Advances in Cryptology-CRYPTO 89; Lecture Notes in Computer Science 435; Springer; 1990

[Yao91] A.C.Yao: "Theory and application of trapdoor functions"; Proceedings of the 23rd IEEE Symposium on Foundation of Computer Science, New York, 1982

Appendix: Some Notations

Throughout the paper n, m and k denote positive integers, and Q and q denote polynomials.

$$
\begin{aligned}
I &= \{0,1\} \\
I^r &= [0 : \lfloor 2^r \rfloor - 1] \quad \text{for} \quad r \in \mathbb{R} \\
F_n &= \{ f \mid f : I^n \to I^n \} \\
\mathcal{F} &= \bigcup_{n>0} F_n \\
B_n &= \{ f \mid f : I^n \to I^n \} \\
\mathcal{B} &= \bigcup_{n>0} B_n \\
S_{2^n} &= \{ \pi \mid \pi \text{ permutes } [0 : 2^n - 1] \} \\
\mathcal{S} &= \bigcup_{n>0} S_{2^n} \\
\mathcal{H} &= \text{a pseudo-random Boolean function generator} \\
l(n) &= \log(2^{n-1}(2n-1)) \\
(x)^{\underline{k}} &= x \cdot (x-1) \cdot \ldots \cdot (x-k+1) \\
G_{l,k} &= \frac{(2^l)^{\underline{k}}}{(2^l)^k}
\end{aligned}
$$

For all $\quad 0 \le i_1 < i_2 < \ldots < i_k < 2^n \quad$ define:

$$
\begin{aligned}
\pi_\omega &= \begin{cases} N(\omega) & \omega \in \Omega_{n,1} (= F_{l(n)}) \\ \omega & \omega \in \Omega_{n,2} (= S_{2^n}) \end{cases} \\
X_n^{\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]}(\omega) &= \left[\begin{smallmatrix} o_1 \\ \vdots \\ o_k \end{smallmatrix}\right] \quad \text{mit:} \quad o_j = \pi_\omega(i_j) \\
L1_n^{\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]} &= \{ \omega \mid \pi_{\omega_l}(\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]) \quad \text{fulfils the LDC} \} \\
L2_n^{\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]} &= \{ \omega \mid \pi_\omega(\left[\begin{smallmatrix} i_1 \\ \vdots \\ i_k \end{smallmatrix}\right]) \quad \text{fulfils the LDC} \}
\end{aligned}
$$