Bitblock Ciphers, Feistel, and SP Networks

Klaus Pommerening
Fachbereich Physik, Mathematik, Informatik
der Johannes-Gutenberg-Universität
Saarstraße 21
D-55099 Mainz

April 7, 1997—English version August 15, 2014
last change January 20, 2021

This chapter contains basic facts about bitblock ciphers and some approaches to their construction.

1 Bitblock Ciphers—Introduction

Description

Bitblock ciphers operate over the alphabet $\Sigma = \mathbb{F}_2 = \{0, 1\}$, and basically encrypt blocks of fixed length conserving this length, controlled by a key that itself is a bitblock of a certain length l. The encryption functions are defined as maps of the set $\mathbb{F}_2^n$ into itself, and the set $\mathbb{F}_2^l$ serves as key space.

For constructing and analyzing bitblock ciphers we usually view $\mathbb{F}_2^n$ as a vector space of dimension n over the two-element field $\mathbb{F}_2$. Sometimes we equip $\mathbb{F}_2^n$ with the structure of the field $\mathbb{F}_{2^n}$, on rare occasions we structure it as cyclic group of order 2^n, thinking of integer addition "with carry" mod 2^n.

Thus we describe a bitblock cipher as a map

$$F \colon \mathbb{F}_2^n \times \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^n$$

or as a family $(F_k)_{k \in K}$ of maps

$$F_k \colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n \quad \text{for } k \in K = \mathbb{F}_2^l$$

where $F_k(a) = F(a, k)$.

Note In this chapter the mathematical symbol n is used for the length of the bitblocks, not for the size of the alphabet.

We might also view a bitblock cipher as a monoalphabetic substitution over the alphabet $\Sigma' = \mathbb{F}_2^n$.

The extension of a cipher to bitstrings of arbitrary lengths is subject of Chapter 3 on "modes", and is of no concern for the moment, and likewise we don't care how to pad shorter bitstrings.

Choice of the Block Length

The block length should be large enough to preclude the methods that break monoalphabetic substitutions, in particular analyses of patterns and frequencies. Moreover we would like to avoid any kind of leaks that reveal information about the plaintext, for example repeated ciphertext blocks.

If the sender didn't systematically prevent repetitions, an attacker could mount a **codebook attack** by collecting pairs of ciphertext and known plaintext for a fixed (but unknown) key. In this way she would construct her own codebook. A large codebook would allow breaking many future messages even if it didn't reveal the key. To prevent this attack we require:

- $\#\Sigma' = 2^n$ should be larger than the number of available memory cells, even assuming a very powerful attacker.

- Keys should be changed quite regularly.

In view of the Birthday Paradox an even stronger criterion is adequate: If the attacker has collected in her codebook about $\sqrt{\#\Sigma'} = 2^{n/2}$ plaintext-ciphertext pairs, the probability of a "collision" is approximately $\frac{1}{2}$. Therefore we require that the number $2^{n/2}$ surpasses the available storage. And keys should be changed long before this number of blocks is encrypted.

In the "pre-AES" age bitblock ciphers usually had 64-bit blocks. From our point of view this is by far insufficient, at best justified by frequent key changes. We prefer 128 bits as block length. This is also the block length of the new standard AES.

This consideration might look somewhat paranoid. But it is a typical example of the security measures in modern cryptography: The cipher designers work with large security margins and avoid any weaknesses even far away from a practical use by an attacker. Thus the security requirements of modern cryptography by far surpass the requirements typical for classical cryptography. This may sound exaggerated. But the modern algorithms—that in fact offer these huge security margins—can also resist future progress of cryptanalytic capabilities.

2 Polynomials over Finite Fields

In this section bitblock cryptography is "reduced" to algebra with polynomials.

Let K be a field. Given a polynomial $\varphi \in K[T_1, \ldots, T_n]$ in n indeterminates $T_1, \ldots, T_n$, we define a function $F_\varphi : K^n \longrightarrow K$ by evaluating the polynomial φ at n-tuples $(x_1, \ldots, x_n) \in K^n$,

$$F_\varphi(x_1, \ldots, x_n) := \varphi(x_1, \ldots, x_n).$$

Note that we carefully distinguish between polynomials and polynomial functions. Polynomials are elements of the polynomial ring $K[T_1, \ldots, T_n]$ where the elements T_i—the "indeterminates"—are a set of algebraically independent elements. That means that the infinitely many monomials $T_1^{e_1} \cdots T_n^{e_n}$ are linearly independent over K.

In general (for infinite fields) there are many more ("non-polynomial") functions on K^n. But not so for finite fields—in other words, over a finite field all functions are polynoms:

Theorem 1 *Let K be a finite field with q elements, and $n \in \mathbb{N}$. Then every function $F : K^n \longrightarrow K$ is given by a polynomial $\varphi \in K[T_1, \ldots, T_n]$ of partial degree $\leq q - 1$ in each T_i.*

The proof of Theorem 1 is in Appendix B, a more elementary proof for the case $K = \mathbb{F}_2$ is in Appendix C.

Corollary 1 *Let $m, n \in \mathbb{N}$. Then every map $F : K^n \longrightarrow K^m$ is given by an m-tuple $(\varphi_1, \ldots, \varphi_m)$ of polynomials $\varphi_i \in K[T_1, \ldots, T_n]$ of partial degree $\leq q - 1$ in each T_i.*

Corollary 2 *Every map $F : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ is given by an m-tuple $(\varphi_1, \ldots, \varphi_m)$ of polynomials $\varphi_i \in \mathbb{F}_2[T_1, \ldots, T_n]$ all of whose partial degrees are ≤ 1.*

From this the **algebraic normal form (ANF)** of a Boolean function $F : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ derives: For a subset $I = \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$ let x^I be the monomial

$$x^I = x_{i_1} \cdots x_{i_r}.$$

Then F has a unique representation as

$$F(x_1, \ldots, x_n) = \prod_I a_I x^I \quad \text{for all } x = (x_1, \ldots, x_n) \in K^n \text{ where } a_I = 0 \text{ or } 1.$$

In particular the 2^n monomial functions $x \mapsto x^I$ constitute a basis of the vector space $\mathrm{Map}(\mathbb{F}_2^n, \mathbb{F}_2)$ over $\mathbb{F}_2$, and the number of these functions is 2^{2^n}.

3 Algebraic Cryptanalysis

Attacks with Known Plaintext

Consider a bitblock cipher, given by the map

$$F\colon \mathbb{F}_2^n \times \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^n$$

Then F is an n-tuple $F = (F_1, \ldots, F_n)$ of polynomial functions in $n + l$ variables all of whose partial degrees are ≤ 1.

An attack with known plaintext $a \in \mathbb{F}_2^n$ and corresponding ciphertext $c \in \mathbb{F}_2^n$ leads to a system

$$F(a, x) = c$$

of n polynomial equations for the unknown key $x \in \mathbb{F}_2^l$.

Systems of polynomial equations (over arbitrary fields) are one of the subjects of algebraic geometry. A rule of thumb says

The solution set for x has dimension 0 *"in general" for $n \geq l$.*

(I. e. it consists of a few isolated solutions. Otherwise, if the solution set allows for free parameters—or has dimension ≥ 1—, the attacker needs some more blocks of known plaintext.)

The general theory of polynomial equations is quite deep, in particular if we search for concrete solution procedures. But maybe the observation that only partial degrees ≤ 1 occur makes a difference?

Examples

Example 1: Let $n = l = 2$,

$$F(T_1, T_2, X_1, X_2) = (T_1 + T_2X_1, T_2 + T_1X_2 + X_1X_2),$$

$a = (0, 1)$, $c = (1, 1) \in \mathbb{F}_2^2$. Then the system of equations for the key $(x_1, x_2) \in \mathbb{F}_2^2$ is

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 + x_1 \\ 1 + 0 + x_1x_2 \end{pmatrix}.$$

The obvious solution is $x_1 = 1$, $x_2 = 0$.

Example 2, linear maps: If F is a *linear* map, then the system of equations has an efficient solution by the methods of linear algebra (n linear equations in l unknowns). For this method to work F needs to be linear only in x.

Example 3, substitution: The complexity (or simplicity) of a polynomial equation is not always clear at first sight. Here is an example (over $\mathbb{F}_2$):

$$x_1x_2x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_2 + x_3 = 0.$$

The substitutions $x_i = u_i + 1$ transform it to

$$u_1u_2u_3 + u_1 = 0$$

(for an easy proof look in the reverse direction). The solutions are

$$u_1 = 0,\ u_2, u_3 \text{ arbitrary } \textit{or } u_1 = u_2 = u_3 = 1.$$

Thus the complete solution of the original equation is

$$x_1 = 1,\ x_2, x_3 \text{ arbitrary } \textit{or } x_1 = x_2 = x_3 = 0.$$

The Complexity of the Algebraic Attack

In the examples the solutions were easily found. But in general this task is too complex.

There are two powerful general approaches for solving systems of (polynomial) equations over $\mathbb{F}_2$:

- SAT solvers. SAT denotes the satisfiability problem of propositional logic. Consider a logical expression in Boolean variables $x_1, \ldots, x_n$ and ask if there exist values of the variables that make the expression "True". In other words consider a Boolean function f and ask if it assumes the value 1. A SAT solver is an algorithm that takes a logical expression and decides the satisfiability by finding a solution x, or showing there's no solution. The naive algorithm uses the truth table and exhausts the 2^n possible arguments. However there are much faster algorithms, the most popular being the DPLL algorithm (after Davis, Putnam, Logemann, and Loveland) and BDD based algorithms (Binary Decision Diagram).
- Elimination using Groebner bases.

Both methods work well for a small number of unknowns. With a growing number of unknowns their complexity becomes unmanageable. Of course we always find a solution by searching through the complete value table. But this naive method is inefficient (exponential in the number of unknowns, hopeless for 80 or more unknowns). But also the costs of SAT solvers and Groebner-basis methods grow exponentially with the number of unknowns. Not even the fact that all partial degrees are ≤ 1 is of vital help. The basic resault is:

Theorem 2 (Garey/Johnson**)** *The problem of finding a common zero of a system of polynoms $f_1, \ldots, f_r \in \mathbb{F}_2[T_1, \ldots T_n]$ is* **NP***-complete.*

Proof. See [1]. ◇

What "**NP**-complete" means will be answered later in this lecture (see Part III, Chapter 6). In fact SAT was the first problem in history shown to be **NP**-complete.

Interpretation

A common interpretation of this theorem is: For an appropriately chosen block cipher $F\colon \mathbb{F}_2^n \times \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^n$ the attack with known plaintext (against the key $k \in \mathbb{F}_2^l$) is not efficient. However from a strict mathematical point of view the theorem *doesn't prove anything* of practical relevance:

1. It relates to an algorithm for *arbitrary* polynomial equations (over $\mathbb{F}_2$). It doesn't contain any assertion for special classes of polynomials, or for a concrete system of equations.

2. It gives a pure proof of (non-) existence, and provides no hint as how to construct a concrete example of a "difficult" system of equations. Note that we know that some concrete systems admit easy solutions.

3. Even if we could find concrete examples of "difficult" systems the theorem would not make any assertion whether only some rare instances (the "worst cases") are difficult, or almost all (the "generic cases")—and this is what the cryptologist wants to know. Maybe there is an algorithm that solves polynomial systems for almost all tuples of unknowns in an efficient way, and only fails for a few exceptional tuples.

Despite these critical comments the theorem raises hope that there are "secure" bitblock ciphers, and the designers of bitblock ciphers follow the

Rule of thumb *Systems of linear equations for bits admit very efficient solutions. Systems of nonlinear equations for bits in almost all cases admit no efficient solution.*

A recent article on the difficulty of systems of polynom equations is

- D. Castro, M. Giusti, J. Heintz, G. Matera, L. M. Pardo: The hardness of polynomial equation solving. Found. Comput. Math. 3 (2003), 347–420.

Interpolation Attack

A variant of algebraic cryptanalysis with known plaintext is the interpolation attack, developed in

- Thomas Jakobsen, Lars R. Knudsen: The interpolation attack on block ciphers, FSE 1997.

The idea is simple: Equip the vector space $\mathbb{F}_2^n$ with a suitable multiplication and interpret it as the finite field $K = \mathbb{F}_{2^n}$ of characteristic 2. An encryption function with a fixed key $k \in \mathbb{F}_2^l$ then is a function $F_k : K \longrightarrow K$, hence a polynomial in one indeterminate over K. Let d be its degree. Then using

interpolation this polynomial is determined by $d+1$ known plaintext blocks. The same is true for the inverse function. Using this the attacker can encrypt and decrypt without knowing the key explicitly.

The cipher designer who wants to prevent this attack should take care that encryption and decryption functions for every fixed key have large degrees as polynomials over K. This is realistic since polynomials over K may have (effective) degrees up to $2^n - 1$.

But beware that this attack may also work for some polynomials of high degree, for example for "sparse" polynomials having only a few coefficients $\neq 0$.

Linearisation of Overdetermined Systems of Equations

Systems of equations of higher order are sometimes solvable, if they are overdetermined, consisting of much more equations than unknowns. Then one simply treats some monomials as additional independent unknowns. Let's illustrate this by a simple example of three equations with two unknowns x and y:

$$\begin{aligned} x^3 + xy + y^5 &= 1, \\ 2x^3 - xy &= 0, \\ xy + 3y^5 &= 3. \end{aligned}$$

We substitute all occuring monomials: $u := x^3$, $v := xy$, $w := y^5$ and get the linear system

$$\begin{aligned} u + v + w &= 1 \\ 2u - v &= 0 \\ v + 3w &= 3 \end{aligned}$$

consisting of three equations involving three unknowns. The solution (in this case even manually derived) is $u = 0$, $v = 0$, $w = 1$. It is unique over a field K of characteristic $\neq 7$. From here we get the complete solution of the original system: $x = 0$, $y = 1$ or any 5th root of unity of K.

This attack gained some popularity in 2002 when there was a rumor that the new AES be vulnerable under this attack. However this rumor didn't survive a closer examination. As it turned out there were much too many dependencies between the linear equations.

4 SP Networks

In an ideal world we would know how to reliably measure the security of a bitblock cipher

$$F\colon \mathbb{F}_2^n \times \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^n$$

for realistic values of the block length n and the key length l, say of an order of magnitude of 128 bits or more.

In fact we know explicit measures of security, for example the linear potential, or the differential potential, that quantify the deviation from linearity, or the algebraic immunity, or others. Unfortunately all of these only give necessary, not sufficient, conditions for security, and moreover the efficient computability of these measures is limited to small block lengths n, about 8 or slightly larger.

Lacking a general efficient approach to security the design of bitblock ciphers usually relies on a structure that, although not obligatory, in practice seems to provide plausible security according to verifiable criteria. Most of the generally approved standard ciphers, such as DES and AES, follow this approach.

Rounds of Bitblock Ciphers

This common design scheme starts by constructing Boolean maps of small dimensions and then extending them to the desired block length in several steps:

1. Define one or more Boolean maps of small dimension q (= block length of the definition domain), say $q = 4$, 6, or 8, that are good for several security criteria. These maps are called **S-boxes** ("S" stands for Substitution), and are the elementary building blocks of the cipher.

2. Mix the round input with some of the key bits and then apply m S-boxes in parallel (or apply the one S-box m times in parallel) to get a map with the desired input width $n = mq$.

3. Then permute the complete resulting bitblock over its total width.

4. These steps together are a **"round"** of the complete scheme. Asset the weaknesses of the round map, that mainly result from using S-boxes of small dimension. Then reduce these weaknesses in a reasonably controlled way by iterating the scheme over several rounds of the same structure but with a changing choice of key bits.

5. Don't stop as soon as the security measures give satisfying values but add some surplus rounds to get a wide security margin.

Figure 1 outlines the scheme for a single round.

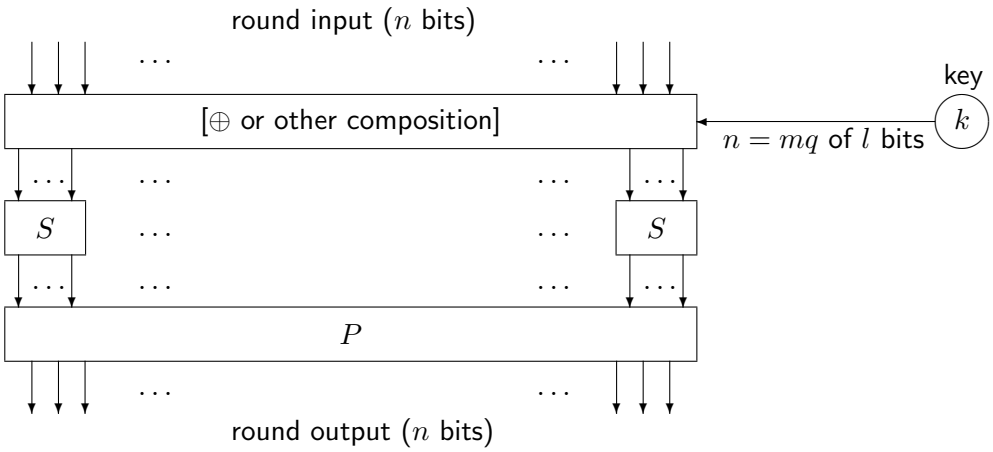


Figure 1: A single round of a bitblock cipher (S is a, maybe varying, S-box, P, a permutation, k, the key)

Shannon's Design Principles

The complete scheme is a special case of a somewhat more general proposal that goes back to SHANNON who required two basic features of block ciphers:

Diffusion The bits of the plaintext block "smear" over all parts of the block. This is done by applying permutations (a. k. a. as transpositions).

Confusion (complex dependencies) The interrelation between plaintext block and key on the one hand, as well as ciphertext block on the other hand should be as complex as possible (in particular as nonlinear as possible). Basic building blocks for this are substitutions.

The overall effect of both requirements, taken together, should result in an unforeseeable change of ciphertext bits for a slight change of the key.

> *The attacker should have no means to recognize whether a guessed key is "nearly correct".*

Product Ciphers after Shannon

For the construction of strong block ciphers SHANNON proposed an alternating sequence of **S**ubstitutions and transpositions (= **P**ermutations), so-called **SP-networks**:

$$\mathbb{F}_2^n \xrightarrow{S_1(\bullet,k)} \mathbb{F}_2^n \xrightarrow{P_1(\bullet,k)} \mathbb{F}_2^n \longrightarrow \dots$$
$$\dots \longrightarrow \mathbb{F}_2^n \xrightarrow{S_r(\bullet,k)} \mathbb{F}_2^n \xrightarrow{P_r(\bullet,k)} \mathbb{F}_2^n$$

depending on a key $k \in \mathbb{F}_2^l$. In this scheme

$$\begin{aligned} S_i &= i\text{-th substitution} \\ P_i &= i\text{-th permutation} \\ P_i \circ S_i &= i\text{-th } \textbf{round} \end{aligned}$$

Alltogether the encryption function consists of r rounds.

Example: Lucifer I (Feistel 1973)

Note that the permutations are special linear maps $P\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$. Some recent bitblock ciphers, the most prominent being AES, replace permutations by more general linear maps that provide an even better diffusion. However the proper term **"LP-network"** is not yet in use.

5 Feistel Networks

Horst Feistel was the first (in the open world) who explicitly applied Shannon's design principles when he constructed the Lucifer ciphers.

The Kernel Map

Assume the blocksize is even: $n = 2s$. Decompose blocks $a \in \mathbb{F}_2^n$ into their left and right halves:

$$a = (L, R) \in \mathbb{F}_2^s \times \mathbb{F}_2^s$$

(We use uppercase letters to avoid confusion with the dimension l of the keyspace.) Moreover we have to agree on the order of the bits in a block:

- The **natural order** has the LSB (Least Significant Bit) always at the right end and assigns it the index 0, the MSB (Most Significant Bit) at the left end with index $n - 1$:

  $$b = (b_{n-1}, \ldots, b_0) \in \mathbb{F}_2^n.$$

  This corresponds to the base 2 representation of natural numbers in the integer interval $[0 \ldots 2^n[$:

  $$b_{n-1} \cdot 2^{n-1} + \cdots + b_1 \cdot 2 + b_0 \in \mathbb{N}$$

  This is the order we use in most situations.

- The **IBM order** has the bits in reverse (LSB at left, MSB at right) and assigns them the indices 1 to n:

  $$a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n.$$

  This corresponds to the usual indexing of the components of a vector. Sometimes, in exceptional cases, the indices 0 to $n - 1$ are used.

The elemantery building blocks of a Feistel cipher are represented by a **kernel map**

$$f \colon \mathbb{F}_2^s \times \mathbb{F}_2^q \longrightarrow \mathbb{F}_2^s,$$

that need not fulfill any further formal requirements. In particular we don't require that the $f(\bullet, k)$ be bijective.

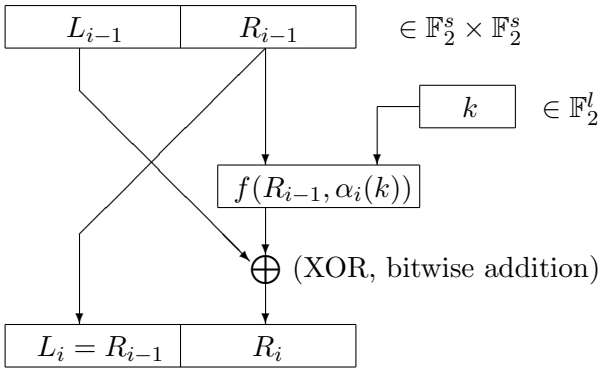
> However to get a useful cipher we should choose a kernel map f that already provides good confusion and diffusion. It should consist of a composition of substitutions and transpositions and be highly nonlinear.

Description of the Rounds

A Feistel cipher consists or r rounds. Each round uses a q-bit round key that is derived from the key $k \in \mathbb{F}_2^l$ by a process called the **key schedule**:

$$\alpha_i \colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^q \quad \text{for } i = 1, \ldots, r.$$

Then round i has this form:



We recognize the autokey principle in form of the addition of the left half and the transformed right half of a bitblock.

Algorithmic Description

From the graphical description we easily derive an algorithmic description:

$$\begin{array}{rcccl}
\textbf{Input} & \longrightarrow & a & = & (a_0, a_1) \in \mathbb{F}_2^s \times \mathbb{F}_2^s \\
 & & a_2 & := & a_0 + f(a_1, \alpha_1(k)) \\
 & & & & \quad \text{– 1st round, result } (a_1, a_2) \\
 & & \vdots & & \vdots \\
 & & a_{i+1} & := & a_{i-1} + f(a_i, \alpha_i(k)) \\
 & & & & \quad \text{– } i\text{-th round, result } (a_i, a_{i+1}) \\
 & & & & \quad \text{– } [a_i = R_{i-1} = L_i,\ a_{i+1} = R_i] \\
 & & \vdots & & \vdots \\
\textbf{Output} & \longleftarrow & c & = & (a_r, a_{r+1}) =: F(a, k)
\end{array}$$

Decryption

The decryption is done by the formula

$$a_{i-1} = a_{i+1} + f(a_i, \alpha_i(k)) \quad \text{for } i = 1, \ldots, r.$$

This boils down to the same algorithm, but the rounds in reverse order. Or in other words: The key schedule follows the reverse direction.

In particular we proved:

Theorem 3 (FEISTEL**)** *Let $F\colon \mathbb{F}_2^{2s} \times \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^{2s}$ be the block cipher with kernel map $f\colon \mathbb{F}_2^s \times \mathbb{F}_2^q \longrightarrow \mathbb{F}_2^s$ and key schedule $\alpha = (\alpha_1, \ldots, \alpha_r)$, $\alpha_i\colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^q$.*

Then the encryption function $F(\bullet, k)\colon \mathbb{F}_2^{2s} \longrightarrow \mathbb{F}_2^{2s}$ is bijective for every key $k \in \mathbb{F}_2^l$.

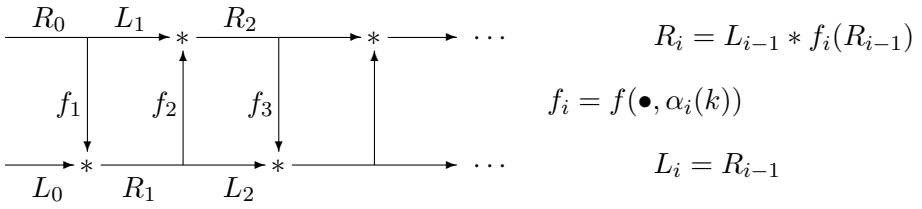
Addendum. *Decryption follows the same algorithm with the same kernel map f but the reverse key schedule $(\alpha_r, \ldots, \alpha_1)$.*

Note When the deryption starts with $c = (a_r, a_{r+1})$, then as a first step *the two halves must be swapped* because the algorithm starts with (a_{r+1}, a_r). To simplify this, in the last round of a FEISTEL cipher the interchange of L and R is usually dropped.

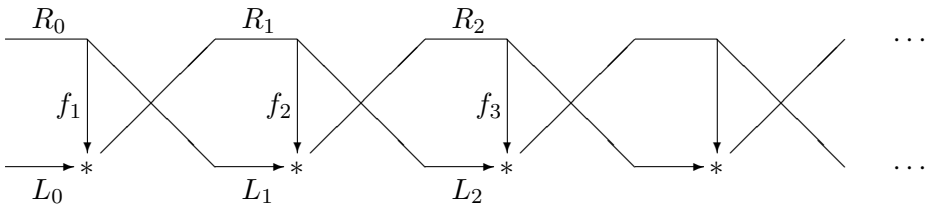
Remarks

- If f and the α_i are linear so is F.
- Usually the α_i are only selections, hence as maps projections $\mathbb{F}_2^l \longrightarrow \mathbb{F}_2^q$.
- Other graphical descriptions of the FEISTEL scheme are:

a) a ladder



b) a twisted ladder



Generalizations

1. Replace the group $(\mathbb{F}_2^s, +)$ by an arbitrary group $(G, *)$. Then the formulas for encryption and decryption are:

$$\begin{aligned} a_{i+1} &= a_{i-1} * f(a_i, \alpha_i(k))), \\ a_{i-1} &= a_{i+1} * f(a_i, \alpha_i(k)))^{-1}. \end{aligned}$$

2. Unbalanced FEISTEL ciphers (SCHNEIER/KELSEY): Divide the blocks into two different halves: $\mathbb{F}_2^n = \mathbb{F}_2^s \times \mathbb{F}_2^t$, $x = (\lambda(x), \rho(x))$. Then the encryption formula is:

$$\begin{aligned} L_i &= \rho(L_{i-1}, R_{i-1}) && \in \mathbb{F}_2^s, \\ R_i &= \lambda(L_{i-1}, R_{i-1}) + f(L_i, \alpha_i(k))) && \in \mathbb{F}_2^t. \end{aligned}$$

Examples

1. LUCIFER II (FEISTEL 1971, published in 1975),
2. DES (COPPERSMITH et al. for IBM in 1974, published as US standard in 1977),
3. many newer bitblock ciphers.

The usefulness of FEISTEL networks relies on the empirical observations:

- By the repeated execution through several rounds the "(s, q)-bit security" (or "local security") of the kernel map f is expanded to "(n, l)-bit security" (or "global security") of the complete FEISTEL cipher F.
- The complete cipher is composed of manageable pieces that may be "locally" optimized for security.

LUBY/RACKOFF underpinned the first of these observations by a theoretical result: A FEISTEL cipher with at least four rounds is not efficiently distinguishable from a random permutation, if its kernel map is random. This means that by FEISTEL's construction a map with good random properties but too small block length expands to a map with good random properties and sufficient block length.

> Michael LUBY, Charles RACKOFF: How to construct pseudorandom permutations from pseudorandom functions. SIAM Journal on Computing 17 (1988), 373–386

Two words of caution about the LUBY/RACKOFF result:

- It doesn't say anything about an attack with known or chosen plaintext.
- It holds for true random kernel maps. However concrete FEISTEL ciphers usually restrict the possible kernel maps to a subset defined by a choice of 2^q keys.

6 Algebraic Attacks for Few Rounds

Formulas for Few Rounds

We write the recursion formula for a FEISTEL cipher as

$$(L_i, R_i) = (R_{i-1}, L_{i-1} + f(R_{i-1}, k_i))$$

where $k_i = \alpha_i(k)$ is the round key.

Proposition 1 *The results (L_r, R_r) of a* FEISTEL *cipher after $r = 2$, 3, or 4 rounds satisfy the equations*

$$\begin{aligned} L_2 - L_0 &= f(R_0, k_1), \\ R_2 - R_0 &= f(L_2, k_2); \end{aligned}$$

$$\begin{aligned} L_3 - R_0 &= f(L_0 + f(R_0, k_1), k_2), \\ R_3 - L_0 &= f(L_3, k_3) + f(R_0, k_1); \end{aligned}$$

$$\begin{aligned} L_4 - L_0 &= f(R_0, k_1) + f(R_4 - f(L_4, k_4), k_3), \\ R_4 - R_0 &= f(L_4, k_4) + f(L_0 + f(R_0, k_1), k_2). \end{aligned}$$

We used minus signs in order to make the formulas valid also for a generalization to abelian groups. In the (present) binary case plus and minus coincide. The purpose of the formulas is that beside the round keys k_i they involve only the plaintext (L_0, R_0) and the ciphertext (L_r, R_r), data that are assumed as known for algebraic cryptanalysis.

Proof. In the case of two rounds the equations are

$$\begin{aligned} L_1 &= R_0, \\ R_1 &= L_0 + f(R_0, k_1), \\ L_2 &= R_1 = L_0 + f(R_0, k_1), \\ R_2 &= L_1 + f(R_1, k_2) = R_0 + f(L_2, k_2); \end{aligned}$$

the assertion follows immediately.

In the case of three rounds we have

$$\begin{aligned} L_1 &= R_0, \\ R_1 &= L_0 + f(R_0, k_1), \\ L_2 &= R_1 = L_0 + f(R_0, k_1), \\ R_2 &= L_1 + f(R_1, k_2) = R_0 + f(L_2, k_2), \\ L_3 &= R_2 = R_0 + f(L_0 + f(R_0, k_1), k_2), \\ R_3 &= L_2 + f(R_2, k_3) = L_0 + f(R_0, k_1) + f(L_3, k_3). \end{aligned}$$

The case of four rounds is left to the reader. ◇

Two-Round Ciphers

For a known plaintext attack assume that L_0, R_0, L_2, R_2 are given. We have to solve the equations

$$\begin{aligned} L_2 - L_0 &= f(R_0, k_1) \\ R_2 - R_0 &= f(L_2, k_2) \end{aligned}$$

for k_1 and k_2. Thus the security of the cipher only depends on the difficulty of inverting the kernel function f. Since usually q, the bitlength of the partial keys, is much smaller than the total key length l the 2^{q+1} evaluations of f for an exhaustion could be feasible. Note that this consideration doesn't depend on the key schedule α—the attacker simply determines the actually used keybits (k_1, k_2).

Example: We equip $\mathbb{F}_2^s$ with the multiplication "$\cdot$" of the field $\mathbb{F}_t$, $t = 2^s$, [see Appendix A] and take

$$f(x, y) = x \cdot y.$$

(Note that f is non-linear as a whole, but linear in the key bits.) Assume the key schedule is defined by $l = 2q$ and k_i = left or right half of k, depending on whether i is odd or even. Then the equations become

$$\begin{aligned} L_2 - L_0 &= R_0 \cdot k_1, \\ R_2 - R_0 &= L_2 \cdot k_2, \end{aligned}$$

hence are easily solved. (If one of the factors R_0 or L_2 vanishes, we need another known plaintext block.)

Of course chosing a kernel map f that is linear in the key bits was a bad idea anyway. But we could solve also slightly more complicated equations, say quadratic, cubic, or quartic.

Three-Round Ciphers

In the case of three rounds the equations are considerably more complex because f is iterated. However the attacker can mount a Meet-in-the-Middle attack with a single known plaintext, if the bit length q of the partial keys is not too large: She calculates the intermediate results (L_1, R_1) of the first round for all possible partial keys k_1, and stores them in a table. Then she performs an exhaustion over the last two rounds as described for two-round ciphers above. The total expenses are $3 \cdot 2^q$ evaluations of f, and 2^q memory cells.

These considerations suggest that FEISTEL *ciphers should have at least four rounds* and support the above mentioned result by LUBY and RACK-OFF. We see how the resistance of the scheme against an algebraic attack increases with the number of rounds, at least if the kernel map f is sufficiently complex.

For the example above with kernel map = multiplication of $\mathbb{F}_{2^s}$ the equations become:

$$\begin{aligned} L_3 - R_0 &= [L_0 + R_0 \cdot k_1)] \cdot k_2, \\ R_3 - L_0 &= [R_0 + R_3] \cdot k_1. \end{aligned}$$

They are nonlinear in the key bits but easily solved in the field $\mathbb{F}_{2^s}$.

Four-Round Ciphers

The equations are much more complex. Even in the example they are quadratic in two unknowns:

$$\begin{aligned} L_4 - L_0 &= [R_0 + R_4 + L_4 \cdot k_2] \cdot k_1, \\ R_4 - R_0 &= [L_4 + L_0 + R_0 \cdot k_1] \cdot k_2. \end{aligned}$$

However in this trivial example they can be solved: eliminating k_1 yields a quadratic equation for k_2 [**Exercise**].

7 Lucifer

History and Relevance

Lucifer was the first published bitblock cipher. Horst Feistel at IBM developed it around 1970. It is in fact a Feistel cipher, and is a predecessor of the standard cipher DES that was developed shortly after. Compared with DES Lucifer seems stronger at first sight, but has some decisive weaknesses that became apparent in the meantime.

> Here we describe the variant that is called "Lucifer II" because of its publication date in 1975. First published (1973 in *Scientific American*) was a somewhat different variant called "Lucifer I".

These are the characteristics of Lucifer (compare the figures below):

- 128-bit key. This is a large enough key space even for today's requirements.
- 128-bit blocks. This is also considered sufficent down to the present day.
- For processing the 128-bit blocks (of keys and texts) are divided into 16 bytes. (From a historic point of we should say "octets" instead of "bytes" since in early computers a byte not necessarily consisted of exactly 8 bits.)
- The Feistel scheme consists of 16 rounds.
- In each round the 8 bytes of the right half R_i of a block (= 64 Bits) are processed quasi in parallel. In other words, every round processes 8 blocks à 1 byte, each one independently from the other ones.
- Each of the rounds consists of a substitution and a permutation. In between some key bits are added (XORed).
- Nonlinearity enters the algorithm only by the substitution. The newly added key bits are processed in a linear way in the actual round, but then are input to the nonlinear substitution of the next round.
- The substitution of a byte starts with a decomposition into two halves à 4 bit each of which is separately transformed by a substitution

  $$S_0, S_1 \colon \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4.$$

  S_0 and S_1 are fixed substitutions used during the whole encryption process. Only the assignement of S_0 and S_1 to the 4-bit halves varies depending on a certain key bit. It is common use to call such nonlinear BOOLEean maps **"S-boxes"** when they occur as even more elementary building blocks of a kernel map.

The algorithm is very well suited for an implementation in hardware, in particular for 8-bit architectures. It is not so efficient in software for its many bit permutations.

Composing the kernel map of many small (identical or similar) S-boxes is a way followed often also today. A *rule of thumb*: the smaller the S-boxes, the more rounds are needed to achieve security.

The presentation of the algorithm follows the paper

- Arthur Sorkin: Lucifer, a cryptographic algorithm. Cryptologia 8 (1984), 22–41.

The Key Schedule

Denote the 16 bytes of the key $k \in \mathbb{F}_2^{128}$ by

$$k = (k_0, \ldots, k_{15}) \in (\mathbb{F}_2^8)^{16}$$

(IBM order but beginning with 0). Round i involves the selection

$$\alpha_i(k) = (\alpha_{ij}(k))_{0 \le j \le 7} \quad \text{of 8 bytes } \alpha_{ij}(k) = k_{7i+j-8 \bmod 16}.$$

This formula looks complicated but describes a quite simple selection scheme:

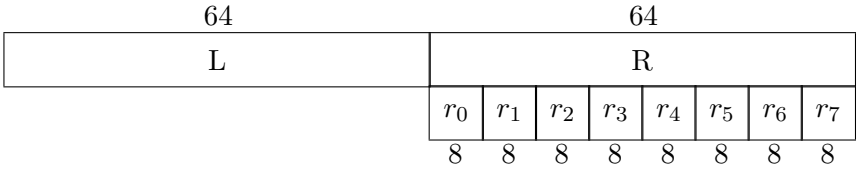
Round	Position							
	0	1	2	3	4	5	6	7
1	0	1	2	3	4	5	6	7
2	7	8	9	10	11	12	13	14
3	14	15	0	1	2	3	4	5
4	5	6	7	8	9	10	11	12
5	12	13	14	15	0	1	2	3
6	3	4	5	6	7	8	9	10
7	10	11	12	13	14	15	0	1
8	1	2	3	4	5	6	7	8
9	8	9	10	11	12	13	14	15
10	15	0	1	2	3	4	5	6
11	6	7	8	9	10	11	12	13
12	13	14	15	0	1	2	3	4
13	4	5	6	7	8	9	10	11
14	11	12	13	14	15	0	1	2
15	2	3	4	5	6	7	8	9
16	9	10	11	12	13	14	15	0

With each new round the selection is cyclically shifted by 7 positions. Note that each byte occurs at each position exactly once. The position defines to which byte of the actual 64-bit blocks the actual key byte applies. Furthermore the byte $\alpha_{i0}(k)$ in position 0 serves as "transformation control byte".

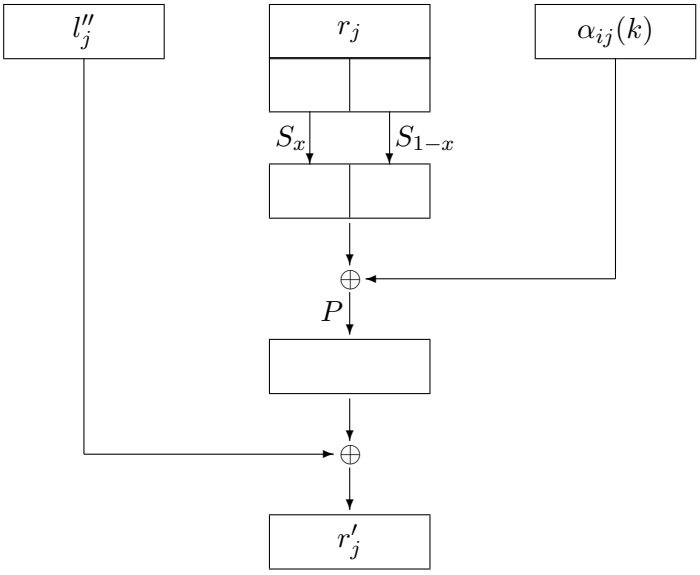
The Round Map

In round i the input, that is the right 64-bit part of the actual block, is divided into eight bytes

$$r = (r_0, \ldots, r_7).$$



Byte number j is transformed as follows by this round:



Here l''_j is a fixed selection of eight bits from the left part of the actual block. The transformation control byte

$$\alpha_{i1}(k) = (b_0, \ldots, b_7)$$

is taken from right to left, and $x = b_{7-j}$.

The definition of the kernel map f is clear from this picture. The explicit formula is not quite compact, therefore we omit it.

The S-Boxes

The S-boxes

$$S_0, S_1 \colon \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$$

are given by their value tables. Here the 4-bit blocks are written in hexadecimal notation, for example $1011 = B$.

$x =$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_0(x) =$	C	F	7	A	E	D	B	0	2	6	3	1	9	4	5	8

$x =$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x) =$	7	2	E	9	3	B	0	4	C	D	1	A	6	F	8	5

The Permutations

The permutation P permutes the bits of a byte as follows:

$$\begin{aligned} P\colon \mathbb{F}_2^8 &\longrightarrow \mathbb{F}_2^8, \\ (z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7) &\mapsto (z_2, z_5, z_4, z_0, z_3, z_1, z_7, z_6). \end{aligned}$$

The bitwise addition of the left half to the transformed right half follows a permutation whose description is: Divide the left half of the actual block into eight bytes:

$$L = (l_0, l_1, l_2, l_3, l_4, l_5, l_6, l_7).$$

Rotate these cyclically after each step. Then for r_j they are in the positions

$$(l'_0, \ldots, l'_7) = (l_j, \ldots, l_{7+j \bmod 8}).$$

Finally construct the byte l''_j as

$$l''_j = (\text{Bit 0 of } l'_7, \text{Bit 1 of } l'_6, \text{Bit 2 of } l'_2, \ldots)$$

etc. in the order $(7, 6, 2, 1, 5, 0, 3, 4)$, and add it to r_j.

References

[1] Michael R. Garey, David S. Johnson, *Computers and Intractability.* Freeman, New York 1979.