Faster Luby-Rackoff Ciphers

Stefan Lucks

Institut für Numerische und Angewandte Mathematik
Georg–August–Universität Göttingen
Lotzestr. 16–18, D–37083 Göttingen, Germany
(email: `lucks@namu01.gwdg.de`)

Abstract. This paper deals with a generalization of Luby's and Rackoff's results [9] on the construction of block ciphers and their consequences for block cipher implementations. Based on dedicated hash functions, block ciphers are proposed which are more efficient and operate on larger blocks than their original Luby-Rackoff counterparts.

1 Introduction

One usually demands a block cipher to be *secure* and *efficient*. Other not-so-common demands are to be *simple* and to *cope with large blocks*.

Consider a cipher being provably secure if another cryptographic primitive is secure. This reduces the number of security assumptions a cryptographic application has to depend on—and thus the application's chance to be broken.

Based on pseudorandom functions, Luby and Rackoff [9] described provably secure block ciphers. This theoretical break-through is also of practical interest—pseudorandomness is a design goal for today's dedicated hash functions. E.g. the authors of RIPE-MD write [14]: "*It is the general view that in order to avoid possible statistical attacks, a good cryptographic function should behave like a random function.*"

Luby-Rackoff ciphers are a special kind of Feistel ciphers. Their difference to "mainstream Feistel ciphers" like DES or TEA is that Luby-Rackoff ciphers require cryptographically stronger round functions than the other ones (i.e. pseudorandom functions), but need less rounds (three or four).

Luby's and Rackoff's construction is *very simple*. At the "Recent Results" session of the 1994 workshop on Fast Software Encryption, Ross Anderson presented Wheeler's and Needham's "Tiny Encryption Algorithm" (TEA) [11]. It is a block cipher and "*short enough to be programmed from memory or a copy.*" Since export, import, possession, and/or use of encryption software is restricted in many parts of the world, a cipher both *secure* and *easy to memorize* would be highly useful. The cipher's *efficiency* would be a concern, too.

Luby-Rackoff ciphers are easy to implement, and there is no need to memorize a complete block cipher—which appears difficult enough, even in the case of TEA. Instead, we only have to memorize a simple framework how to encrypt by the use of an innocent-looking authentication tool like a dedicated hash function.

Often, it is desirable that a small plaintext modification scrambles as many ciphertext bits as possible—a motivation to consider chaining modes (see [17], section 8.1)) and large blocksizes. Also, if data are transmitted or stored in large clusters—imagine sector-wise disk encryption—it makes sense to consider a block cipher whose block size equals the data cluster size.

In this paper, we generalize Luby's and Rackoff's approach. Our ciphers are *easy* to memorize and implement, *provably secure* (if the underlying functions are truly pseudorandom), *efficient*, and *flexible* with respect to the block size.

By "efficient" we focus on the throughput. The key-setup time, the overhead for encrypting unnecessary bits (if the length of the plaintext is no multiple of the block size), and the space required are of minor importance for us.

Without victimizing either the construction's simplicity or the provability of security, we simultaneously increase both the throughput and the blocksize. This appears paradoxical—changing a plaintext bit scrambles one ciphertext block, so one intuitively should expect larger blocks to require "more work to do".

Our ciphers have a lot in common with Blaze's and Schneier's [3] "generalized unbalanced Feistel networks" (GUFNs). Both generalize Feistel ciphers by introducing round functions, where the length of the control block is different from the length of the target block.

Feistel ciphers usually repeat the same round function, changing only the round (sub-)key and swapping the control and the target block. This also seems to be true for GUFNs, like "MacGuffin" [3], though, of course, the target and control block are not swapped, but their bits are subject to a more general permutation. In the case of MacGuffin, the control block consists of 48 bits and the target block of 16 bits. Our ciphers are different: Round functions $f_i : \{0,1\}^r \longrightarrow \{0,1\}^l$ and round functions $f_j : \{0,1\}^l \longrightarrow \{0,1\}^r$ alternate—where $r \neq l$ is possible.

Block ciphers with large blocks have been studied before, e.g. by Kaliski and Robshaw [8]. Their cipher is fast, the blocksize is fixed to 8192 bits, and the design is similar to the design of the MD5 hash function. Kaliski and Robshaw do not relate the security of their cipher to the security of MD5.

A fast cipher with variably-sized blocks is Wheeler's WAKE [18].

There is some similarity between the block ciphers described here and a certain cipher in practical use [5]. As we will see below, that cipher has a subtle security flaw. This stresses the need to scrutinize such constructions.

2 Cryptographic Primitives

In this section, we give a sketchy description of some cryptographic primitives. For their complete definitions see [12], section 4.

In the following, we will not distinguish between a pair $(x, y) \in \{0,1\}^m \times \{0,1\}^n$ of two bit strings $(x_1, \ldots, x_m)$ and $(y_1, \ldots y_n)$ and the concatenated bit string $(x_1, \ldots, x_m, y_1, \ldots, y_n) \in \{0,1\}^{m+n}$.

By *randomly chosen* we always mean "chosen according to the uniform probability distribution". For "x is randomly chosen from the set M" we also write $x \in_{\mathrm{R}} M$. A k-bit *random string* is a randomly chosen k-tuple of bits. Similarly, an n to m bit *random function* (a *random permutation* over $\{0,1\}^n$) is a randomly chosen function $f : \{0,1\}^n \longrightarrow \{0,1\}^m$ (resp. a randomly chosen permutation over $\{0,1\}^n$).

For the notions of *efficient* and *infeasible* we have to consider two standpoints:

- Given families of algorithms or problems indexed by a security parameter n, algorithms are "efficient" if they are executable in polynomial (in n) time, and problems are "infeasible" if there are no efficient algorithms for them.
- When a value for n is chosen—or cryptographic primitives without a security parameter are considered—"efficient" means that we have enough time and space to execute the algorithm and "infeasible" means no enemy has enough time and space to solve the problem.

The model of computation is not important for us—instead of fast algorithms we also could consider small (i.e. polynomial-size) Boolean circuits.

Dealing with large random strings, random functions, or random permutations is normally impractical. Usually one starts with a small randomly chosen input (the "seed") and relies on "pseudorandomness". Note that the size of the seed is a security parameter.

A *bit generator* A is an efficient deterministic algorithm which for $l > n$ computes a function $f : \{0,1\}^n \longrightarrow \{0,1\}^l$.

Let y be an l-bit string with

①	$y \in_{\mathrm{R}} \{0,1\}^l$	with probability 0.5 and
②	$y = f(x), x \in_{\mathrm{R}} \{0,1\}^n$	with probability 0.5.

A is a *pseudorandom bit generator* (*PBG*) if for any efficient algorithm the probability to find out whether y is chosen according to ① or ② is *negligible*, i.e. for any polynomial Q and large enough n the probability is below $Q(n)/2^n$. By iterated use of A we can generate a pseudorandom bit string of any length from an n-bit seed.

It is well known that PBGs exist if and only if one-way functions exist, see [7] and [6].

In short, a bit generator is a function which outputs bit strings, and it is pseudorandom, if it is infeasible to distinguish between the generated random strings and "truly" random strings.

Similarly, a function (permutation) generator is an efficient algorithm which outputs a function (permutation). Such a generator is a *pseudorandom function (permutation) generator*, short *PFG* (*PPG*), if it is infeasible to distinguish between its output and "truly" random functions (permutations). We consider an efficient distinguishing algorithm D to decide if a function f (a permutation p) is randomly chosen or not. D is not given a description of f (the permutation p) as input, but D has access to a "black box" which, given an input x, computes $f(x)$ (resp. $p(x)$) in time 1.

3 How to Construct Cryptographic Primitives

Every PBG f constitutes a secure stream cipher: Use the key as seed and the output of f just like a one-time pad. Any efficient algorithm to break the stream cipher allows to distinguish efficiently between the output of the bit generator and random bit strings—in contradiction to the pseudorandomness of f.

For any pair of algorithms A and A', where A relies on randomness and in A' the random elements of A are replaced by their pseudorandom counterparts, it is infeasible to distinguish between the behaviour of A and A'.

Let $f' : \{0,1\}^{n'} \longrightarrow \{0,1\}^{2n'}$ and $f : \{0,1\}^{n'} \longrightarrow \{0,1\}^{n}$ be PBGs. Either f or f' could be defined by the help of the other one by iterated use or cutting off output bits. Goldreich, Goldwasser and Micali [4] demonstrated how to build an m to n bit PFG, a *GGM-PFG*. Let $K \in_{\mathrm{R}} \{0,1\}^{n'}$ be the seed. The functions f'_0, f'_1 are defined by $f'_0(x) = (y_1, \dots, y_n)$, $f'_1(x) = (y_{n+1}, \dots, y_{2n})$, where $(y_1, \dots, y_{2n}) = f'(x)$. Then the function

$$F_K(x) : \{0,1\}^m \longrightarrow \{0,1\}^n, \quad F_K(x_1, \dots, x_m) = f\big(f'_{x_1}(f'_{x_2}(\dots(f'_{x_m}(K))\dots))\big)$$

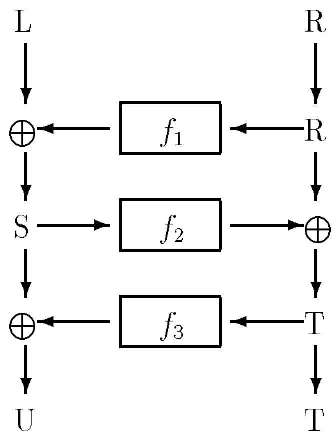
is pseudorandom and thus $F : \{0,1\}^{n'} \times \{0,1\}^m \longrightarrow \{0,1\}^n$ with $F(K,x) = F_K(x)$ is a PFG. Its computation requires to (re-)initialize the PBG f' m times. This slows down the computation of F. Note also that we must generate between $n'm + n$ and $2n'm + n$ pseudorandom bits. For reasonable choices of n' and except when $n \gg m$, the time to compute an m to n bit pseudorandom function is roughly proportional to m.

A PPG F behaves almost like a block cipher. The distinguisher's access to a black box containing the random permutation F_K is a model for an attacker who launches an adaptive chosen message attack. Due to its pseudorandomness, F is secure against such attacks.

When using F as a block cipher, not only the computation of F_K but also the computation of F_K^{-1} must be efficient.

Let f_1, f_2, and f_3 be random functions $f_1, f_3 : \{0,1\}^r \longrightarrow \{0,1\}^l$ and $f_2 : \{0,1\}^l \longrightarrow \{0,1\}^r$. By "$\oplus$" we denote the bit-wise XOR. We compute values $S, U \in \{0,1\}^l$ and $T \in \{0,1\}^r$ by $S = L \oplus f_1(R)$, $T = R \oplus f_2(S)$, and $U = S \oplus f_3(T)$, This way we have defined a permutation $p(L,R) = \psi(f_1, f_2, f_3)(L,R) = (U,T)$ over $\{0,1\}^{l+r}$. This is represented by figure 1—just start with L and R and follow the arrows. Similarly, $\psi(f_3, f_2, f_1)$ gives the inverse permutation.

Luby and Rackoff showed in their famous paper [9] that p is indistinguishable from a random function if $l = r$ and f_1, f_2, and f_3 are random or pseudorandom functions. If a permutation p is undistinguishable from a random function, it also undistinguishable from a random permutation. Maurer [10] gave an amazingly simple proof for Luby's and Rackoff's theorem. Neither [9] nor [10] regarded $l \neq r$, but it is not difficult to generalize Maurer's proof to handle this case. Here, we give the proof and in the sequel deal with some of its arguments.



$$f_1, f_3 : \{0,1\}^r \longrightarrow \{0,1\}^l$$
$$f_2 : \quad \{0,1\}^l \longrightarrow \{0,1\}^r$$

Fig. 1. Permutation $\psi(f_1, f_2, f_3)(L, R) = (U, T)$

Theorem 1. *Let with probability* 0.5 $g : \{0,1\}^{r+l} \longrightarrow \{0,1\}^{r+l}$ *be a random function and with probability* 0.5 $g = \psi(f_1, f_2, f_3)$, *where* $f_1, f_3 : \{0,1\}^r \longrightarrow \{0,1\}^l$ *and* $f_2 : \{0,1\}^l \longrightarrow \{0,1\}^r$ *are random functions. Let* $n \leq \min\{l, r\}$ *be a security parameter.*

Let A be a distinguisher. Given a "black box" which is able to compute g, A outputs either 1 or 0. By P_{RAND} and P_{PERM} we denote the probabilities for A to output 1 if g is randomly chosen, resp. if $g = \psi(f_1, f_2, f_3)$.

A accesses the "black box" at most k times, i.e. A chooses at most k inputs $(L_1, R_1), \ldots, (L_k, R_k)$ and receives the corresponding $(U_1, T_1), \ldots, (U_k, T_k)$ with $(U_i, T_i) = g(L_i, R_i)$. Then

$$|P_{\mathrm{RAND}} - P_{\mathrm{PERM}}| < \frac{k^2}{2^n}. \tag{1}$$

Proof. Let g be no random function. If, for the given inputs, g behaves like a random function, A has no chance to find out that g is none.

Repeating a question does not provide new information for A. We assume $(L_i, R_i) \neq (L_j, R_j)$ for $i \neq j$, i.e. $\left(L_i \neq L_j \text{ or } R_i \neq R_j\right)$.

Given $i \neq j$, denote by $p^{(=)}(S_i, S_j)$ the probability for $S_i = S_j$ and by $p^{(\neq)}(S_1, \ldots, S_k)$ the probability that the $S_1, \ldots, S_k$ are pairwise distinct.

$R_i = R_j \Longrightarrow (L_i \neq L_j$ and $S_i \neq S_j)$. If $R_i \neq R_j$, then $f_1(R_i)$ and $f_1(R_j)$ are two independent l-bit random values—and so are $S_i = f_1(R_i) \oplus L_i$ and $S_j = f_1(R_j) \oplus L_j$. Thus

$$p^{(=)}(S_i, S_j) \leq 2^{-l}. \tag{2}$$

There are $k(k-1)/2$ two-sets $\{i,j\}$, therefore

$$p^{(\neq)}(S_1,\ldots,S_k) \geq 1 - \frac{k(k-1)/2}{2^l} \geq 1 - \frac{k(k-1)/2}{2^n}. \tag{3}$$

Since $(L_i,R_i) \neq (L_j,R_j) \Longleftrightarrow (S_i,R_i) \neq (S_j,R_j)$,

$$p^{(\neq)}(T_1,\ldots,T_k) \geq 1 - \frac{k(k-1)/2}{2^r} \geq 1 - \frac{k(k-1)/2}{2^n}.$$

Denote by p^* the probability that all S_i are pairwise distinct and all T_i are so as well. Combining $p^{(\neq)}(S_1,\ldots,S_k)$ and $p^{(\neq)}(T_1,\ldots,T_k)$ gives

$$p^* \geq 1 - \frac{k(k-1)}{2^n} > 1 - \frac{k^2}{2^n}.$$

If the S_i are pairwise distinct *and* the T_i are pairwise distinct, then the $f_2(S_1),\ldots,f_2(S_k)$ and $f_3(T_1),\ldots,f_3(T_k)$ are all independent random values, and so are the outputs $(U_1,T_1),\ldots,(U_k,T_k)$.

With at least the probability p^*, the function g behaves like a random function. □

If the functions f_1, f_2 and f_3 are not random but pseudorandom, $\psi(f_1,f_2,f_3)$ represents a pseudorandom permutation—and a practical three round Feistel cipher as well. How efficient is it? At least for GGM-PFGs, the time needed to compute an m to n bit pseudorandom function is roughly proportional to m. Then the throughput of our cipher is roughly proportional to $(l+r)/(2r+l)$.

In order to speed-up the cipher, it appears natural to consider $r \ll l$. In the sequel we paradoxically concentrate on $l < r$. Below we will see that the r to l bit (pseudo-)random functions f_1 and f_3 can be replaced by more efficient constructs.

Note that $n \leq \min\{r,l\}$ is a security parameter, hence neither r nor l may be too small—one reason to consider large block sizes.

4 Hash Functions as Building Blocks

GGM-PFGs are not too efficient. A PBG with input size n' is initialized and computes $2n'$ output bits. Then the PBG is initialized again, using either the left or the right half of the output as new key. Finally, after m iterations, the output bits of an m to n bit GGM-PFG can be computed.

Fast stream ciphers typically provide poor efficiency for GGM-PFGs. Either, extensive preprocessing is necessary for every new key—as for Rogaway's and Coppersmith's SEAL [15]. Or the keysize n' is just too large—e.g. in the case of Anderson's PIKE [2].

As an alternative, we consider cryptographic hash functions. In recent years, a large number of them have been published, e.g. MD4, MD5, SHA, HAVAL, and RIPE-MD (see [17]). Their security is unproved, but they are fast and at least *appear* to be secure[1].

For us, there is no need to distinguish between a general hash function $h : \{0,1\}^* \longrightarrow \{0,1\}^o$ and its restriction $h : \{0,1\}^i \longrightarrow \{0,1\}^o$ to i-bit inputs.

Collision intractable hash functions can be too weak for hash function applications (see Anderson [1]). A stronger property is pseudorandomness: A hash function is $h : \{0,1\}^i \longrightarrow \{0,1\}^o$ *pseudorandom*, if for $K \in_{\mathrm{R}} \{0,1\}^i$ (the secret key) the function $F_K : \{0,1\}^i \longrightarrow \{0,1\}^o$ with $F_K(x) = h(K \oplus x)$ is undistinguishable from a random function.

When using hash functions as building blocks for ciphers, we demand the hash functions to be pseudorandom. This seems reasonable since non-pseudorandom dedicated hash functions fail a widely accepted security-related design goal and should not be used anyway.

5 A Possible Speed-up for the First Round

Necessary for the proof of theorem 1 is that with high probability different $(l+r)$-bit inputs lead to different S_i, S_j.

Let $d \geq n \geq m$, where n is a security parameter. The function $F : \{0,1\}^d \times \{0,1\}^r \longrightarrow \{0,1\}^m$ is a *difference concentrator* if $m < r$ and for $K \in_{\mathrm{R}} \{0,1\}^d$, all $x \neq y$ and all c

$$F(K,x) \oplus F(K,y) = c \tag{4}$$

with probability 2^{-n} or less. F is a *computational difference concentrator* if $m < r$ and for $K \in_{\mathrm{R}} \{0,1\}^d$ it is infeasible to find three values $x \neq y$ and c such that (4) holds with nonnegligible probability.

Theorem 2. *Let $F : \{0,1\}^d \times \{0,1\}^r \longrightarrow \{0,1\}^l$ be a difference concentrator and $F' : \{0,1\}^d \times \{0,1\}^r \longrightarrow \{0,1\}^l$ a computational difference concentrator.*

(a) *If the conditions for theorem 1 are given, except for $f_1(R) = F(K,R)$ with $K \in_R \{0,1\}^d$, then bound (1) holds, i.e. $|P_{\mathrm{RAND}} - P_{\mathrm{PERM}}| < \frac{k^2}{2^n}$.*
(b) *If the conditions for theorem 1 are given, except for $f_1(R) = F'(K,R)$ with $K \in_R \{0,1\}^d$, $\psi(f_1, f_2, f_3)$ is a pseudorandom permutation.*

Proof. Consider (a). Let $K \in_{\mathrm{R}} \{0,1\}^d$ be a secret key and $f_i(x) = F(K,x)$. If bound (3) still holds, i.e. if

$$p^{(\neq)}(S_1, \ldots, S_k) \geq 1 - \frac{k(k-1)/2}{2^n},$$

the proof of theorem 1 is still valid. If there are values $S_i = S_j$ for $i \neq j$, then $R_i \neq R_j$ and

$$\begin{array}{rlcl} & f_1(R_i) \oplus L_i & = f_1(R_j) \oplus L_j, & \\ \iff & f_1(R_i) \oplus f_1(R_j) & = L_i \oplus L_j & = c, \\ \iff & F(K,R_i) \oplus F(K,R_j) & = L_i \oplus L_j & = c, \end{array}$$

[1] As an anonymous referee reported, MD4 has been broken recently.

which is possible with probability 2^{-n} for all c, R_i, R_j with $R_i \neq R_j$. Hence $p^{(=)}(S_i, S_j) \leq 2^{-n}$, from which bound (3) follows. Now, (b) is easy. $\square$

Let n be a security parameter and $K = (k_{i,j}) \in \{0,1\}^{n \times r}$ a secret key. We define $F_K : \{0,1\}^r \longrightarrow \{0,1\}^n$ by $F_K(x) = F_K(x_1, \ldots, x_r) = (F_K^1(x), \ldots, F_K^n(x))$ with $F_K^i(x) = 0$ if $x = (0, \ldots, 0)$ and otherwise

$$F_K^i(x_1, \ldots, x_r) = \bigoplus_{j=1}^{r} k_{i,j} x_j.$$

If K is randomly chosen, the function F_K gives a secure replacement for f_1 in the first round (if $n < l$ we simply append $l - n$ zero bits to the output of F_K).

Theorem 3. *The function F with $F(K, x) = F_K(x)$ is a difference concentrator.*

Proof. For all $x, y \in \{0,1\}^r$, we have

$$F_K^i(x) \oplus F_K^i(y) \;=\; \left(\bigoplus_{j=1}^{r} k_{i,j} x_j\right) \oplus \left(\bigoplus_{j=1}^{r} k_{i,j} y_j\right) \;=\; \bigoplus_{j=1}^{r} k_{i,j}(x_j \oplus y_j) \;=\; F_K^i(x \oplus y).$$

If $x \neq y$, then $F_K(x \oplus y)$ is the bit-wise XOR of one or more of the randomly chosen n-bit vectors $((k_{1,1}, \ldots, k_{1,n}), \ \ldots \ (k_{r,1}, \ldots, k_{r,n}))$. Hence $F_K(x \oplus y)$ is a randomly chosen n-bit vector itself. $\square$

Is there a way to make use of a fast hash function h with l output bits? A naive approach to replace $f_1(R)$ by $h(R)$ fails since for random L_1, R_1, and R_2 we can compute $L_2 := L_1 \oplus h(R_1) \oplus h(R_2)$ and get $S_2 = L_2 \oplus h(R_2) = L_1 \oplus h(R_1) = S_1$. But for $K \in_{\mathrm{R}} \{0,1\}^k$, we can replace f_1 by $H_K = h(R \oplus K)$.

Theorem 4. *If the function $h : \{0,1\}^r \longrightarrow \{0,1\}^l$ is pseudorandom, the function H with $H(K, x) = h(x \oplus K)$ is a computational difference concentrator.*

Proof. If f is a random function $\{0,1\}^r \longrightarrow \{0,1\}^l$ and $x \neq y$, $f(x \oplus K)$ and $f(y \oplus K)$ are independent random values. Thus $c^* = f(x \oplus K) \oplus f(y \oplus K)$ is random. The function h is pseudorandom, so $c = h(x \oplus K) \oplus h(y \oplus K)$ is a pseudorandom value. $\square$

6 A "Shortcut" for the Third Round

With respect to the third round, we demand for the proof of theorem 1 that the outputs of function f_3 are all independent random values—when k different inputs are fed into the cipher.

From the rounds before, we expect to see k independent random r-bit values $T_1, \ldots, T_k$ as inputs for f_3. The probability that there is a $i \neq j$ with $T_i = T_j$, is $k^2/2^r$. This is overkill since $r > l \geq n$. We replace f_3 by a random function $f_* : \{0,1\}^l \longrightarrow \{0,1\}^l$, the remaining $r - l$ input bits are ignored. W.l.o.g. we use the rightmost l bits, i.e. we treat $T \in \{0,1\}^r$ like the binary representation of a number $T \in \{0, \ldots, 2^r - 1\}$ and replace $f_3(T)$ by $f_*(T \bmod 2^l)$. It is easy to verify

Theorem 5. *Let the function* $f_* : \{0,1\}^l \longrightarrow \{0,1\}^l$ *be a random function.*

(a) *If—except for* $f_3(T) = f_*(T \bmod 2^l)$*—the conditions of theorems 1 or 2(a) are satisfied, then* $|P_{\text{RAND}} - P_{\text{PERM}}| < \frac{k^2}{2^n}$.

(b) *If—except for* $f_3(T) = f_*(T \bmod 2^l)$*—the conditions of theorems 1, 2(a), or 2(b) are satisfied,* $\psi(f_1, f_2, f_3)$ *is a pseudorandom permutation.*

We might as well use a random function $f_{**} : \{0,1\}^n \longrightarrow \{0,1\}^l$ instead of f_*.

7 How to Define "Security"

Using a pseudorandom function, we did show how to construct a block cipher. We have shown above that if the pseudorandom function is secure, the hash function is secure, too. But what is meant by "secure" in this context?

In theoretical cryptography, the "security" of a scheme often is reduced to the non-existence of probabilistic polynomial time algorithms to break it. Luby's and Rackoff's result is much stronger! Recall the distinguisher A and the probabilities P_{RAND} and P_{PERM} in theorem 1. By P_{PSEU}, we denote the probability that A outputs 1 if A accesses the function $g = \psi(f_1, f_2, f_3)$ with pseudorandom f_i at most k times. If

$$|P_{\text{RAND}} - P_{\text{PSEU}}| \geq p + \frac{k^2}{2^n}$$

holds for $p > 0$, then it is straightforward to use A as a test for the randomness of f_1, f_2, and f_3. The distinguishing probability is at least p:

$$|P_{\text{PERM}} - P_{\text{PSEU}}| \geq p.$$

In other words, attacks on Luby-Rackoff ciphers are at least as hard as attacks on the underlying pseudorandom function generators, except for possibly increasing the attacks' probability of success by $k^2/2^n$:

> *Let* $g = \psi(f_1, f_2, f_3)$ *be an encryption function. Let* f_1*,* f_2*, and* f_3 *be generated by a pseudorandom function generator which is secure in the following sense:*
>
> > *There is no algorithm which in time t (i.e. the time required to encrypt t plaintexts) distinguishes between random and pseudorandom* f_1*,* f_2*, and* f_3 *with probability p or more.*
>
> *Then the block cipher defined by g is secure in the following sense:*
>
> > *There is no algorithm A to distinguish between g and a random function in time t with probability* $p + k^2/2^n$ *or more, where A chooses exactly k inputs* $x_1, x_2, \ldots, x_k$ *and gets the corresponding outputs* $g(x_1), g(x_2), \ldots, g(x_k)$.

Note that the computation of the k ciphertexts takes time k and is a part of A's overall run time. Also note that in the case of both theorems 2(a) and 5(a) bound (1)

$$|P_{\text{RAND}} - P_{\text{PERM}}| < \frac{k^2}{2^n}.$$

holds, so neither the use of computational difference concentrators nor the third round "shortcut" does reduce the block cipher's security—in the above sense.

8 Implementation Considerations

Our constructions seem to require a large number of key bits. We anticipate that all key bits are generated pseudorandomly, given a short initial key.

Difference Concentrators

Theorem 3 deals with a difference concentrator F. Its implementation requires rn bits of storage space for the key matrix. A naive implementation of F would require about $\frac{2nr}{w}$ operations on w-bit words. Table-look-up techniques seem to enable a reasonable fast computation of F at the cost of additional storage space.

A possibly more practical realization of difference concentrators may be based on irreducible CRC generator polynomials of prime degree, cf. [13]. Note that in this case the CRC generator polynomial is part of the secret key.

Dedicated hash functions are well suited to act as computational difference concentrators.

GGM-PFG Based Implementations

How do the above modifications of the basic Luby-Rackoff scheme effect the throughput? Table 1 gives some estimations. We measure the throughput as

$$\text{throughput} = \frac{\text{number of encrypted bits}}{\text{number of PBG initializations}}$$

and compare it to $S_{\text{L\&R}}$, the throughput of Luby's and Rackoff's original proposal with $l = r$.

If $l < r$, the function f_3 can either be realized as an r to l bit GGM-PFG or, according to theorem 5, as an l to l bit PFG. This is indicated by (**).

With respect to table 1, the function f_2 is realized as an l to r bit GGM-PFG, and f_1 can either be realized as r to l bit GGM-PFG or as a (computational) difference concentrator, see theorem 2, indicated by (*) in the table.

case	$l = r$	$l > r$	$l < r$	$l < r$	$l < r$	$l < r$
f_1	GGM	GGM	GGM	(*)	GGM	(*)
f_3	r to l	r to l	r to l	r to l	(**)	(**)
through-put	$\frac{2}{3}(r+l)$ $= S_{\text{L\&R}}$	$\frac{r+l}{2r+l}$ $> S_{\text{L\&R}}$	$\frac{r+l}{2r+l}$ $< S_{\text{L\&R}}$	$1-$const $> S_{\text{L\&R}}?$	$\frac{r+l}{2r+l}$ $> S_{\text{L\&R}}$	$\frac{r+l}{2l}-$const $\gg S_{\text{L\&R}}?$

Table 1. Performance of GGM-PFG based implementation of $\psi(f_1, f_2, f_3)$.

Based on Hash Functions

At present, dedicated hash functions seem to be more practical for PFGs than GGM-PFGs, cf. section 4.

In the sequel, one block consists of b subblocks of size l, thus $r = l(b-1)$. We divide the bit strings $R, T \in \{0,1\}^r$ into subblocks $R_{(1)}, \ldots R_{(b-1)}$ and

$T_{(1)}, \ldots T_{(b-1)}$, each of size l. The keys are

$$\begin{array}{ll} K_1 & \in_R \{0,1\}^r \\ K_2 = (K_{2,1}, \ldots, K_{2,b-1}) & \in_R \{0,1\}^r \\ K_3 & \in_R \{0,1\}^l \end{array}$$

With respect to Theorem 5 and based on a hash function h with l output bits, we propose a general framework:

$$\begin{array}{lcl} S & = & L \oplus h(K_1 \oplus R) \\ T_{(i)} & = & R_{(i)} \oplus h(K_{2,i} \oplus S) \quad (1 \leq i < b) \\ U & = & S \oplus h(K_3 \oplus T_{(1)}) \end{array} \tag{5}$$

In the third equation of (5), we can replace $T_{(1)}$ by $T_{(i)}$ for any $i \in \{1, \ldots, b-1\}$.

If h is a pseudorandom function, then the equations $T_{(i)} = R_{(i)} \oplus h(K_{2,i} \oplus S)$ also represent an l to r bit pseudorandom function. We might as well realize it by *cipher feedback* (*CFB*), similar to Peter Gutmann's "message digest cipher" (MDC) [5]:

$$\begin{array}{lcl} T_{(1)} & = & R_{(1)} \oplus h(K_2 \oplus S) \\ T_{(i)} & = & R_{(i)} \oplus h(K_2 \oplus T_{(i-1)}) \quad (1 < i < b) \end{array} \tag{6}$$

Often CFB is used to encrypt the plaintext $R = \left(R_{(1)}, \ldots, R_{(b-1)}\right)$ to the ciphertext $T = \left(T_{(1)}, \ldots, T_{(b-1)}\right)$—depending on the key $K_2 \in_R \{0,1\}^l$ and the *initial value* (*IV*) S. The security of CFB encryption does not depend on keeping S secret.

CFB copes well with block sizes not divisibly by l. The unnecessary bits of $T_{(b-1)}$ can be ignored without consequences for the decryption process.

Note that CFB is inherently sequential—we can't compute the $T_{(i)}$ in parallel. This is a disadvantage, compared to (5). Some general problems with CFB are:

- If K_2 and S remain constant, changing the i-th plaintext subblock $R_{(i)}$ to $R^*_{(i)}$ does not affect the first $i-1$ ciphertext blocks.
- Worse, if $T_{(i)}$ and $T^*_{(i)}$ are the ciphertext blocks corresponding to $R_{(i)}$ and $R^*_{(i)}$, then $T_{(i)} \oplus T^*_{(i)} = R_{(i)} \oplus R^*_{(i)}$.

These problems can be overcome by using different IVs for each different input. This is exactly what we achieve by the use of difference concentrators.

9 The Encryption Function of SFS

We have seen above that difference concentrators—like the one defined by the first equation of (5)—are sufficient for secure Luby-Rackoff like block ciphers. What, if we replace a difference concentrator by something simpler and more efficient? The little piece of cryptanalysis below demonstrates what may go wrong.

Peter Gutmann's [5] SFS ("Secure FileSystem") is a popular tool for the sector-wise encryption of data to be stored on disk. In this section we consider the encryption function of SFS, which works as described in (6) and is based on the hash function SHA. SHA generates 160 output bits.

A sector is represented as a 4096-bit plaintext block, split up into 32-bit words $P_0, P_1, \ldots, P_{127}$. Each sector has a unique "sector-IV", a 160-bit value $P_{-5}, \ldots, P_{-1}$. The IV itself depends on the sector-IV. For different Sectors, all IVs are likely to be different. We may assume the sector-IVs to be secret.

We worry about enemies reading our storage medium twice—*before* and *after* we have modified the data—or reading backups of different age.

To get different IVs each time a different plaintext block is encrypted, SFS uses a "word-wise scrambler polynomial" [5]:

$$P_i := P_i \oplus P_{i-4} \oplus P_{i-5},$$

starting with $i = 0$. The last five words $P_{123}, \ldots, P_{127}$ are used as IV.

The i-th bit $P_j^{(i)}$ in P_j only affects the five bits $P_{123}^{(i)}, \ldots, P_{127}^{(i)}$ in the IV. In fact, SFS generates 32 independent partial checksums in parallel. Each plaintext bit affects exactly one partial checksum.

Attackers can easily compute how to modify a file (or a sector-sized part of it) without changing the IV. If $i \in \{0, \ldots, 122\}$, inverting the j-th bits $P_i^{(j)}$, $P_{i+4}^{(j)}$ and $P_{i+5}^{(j)}$ of P_i, P_{i+4}, and P_{i+5} does not affect the corresponding checksum bits $P_{123}^{(j)}, \ldots, P_{127}^{(j)}$, whatever the sector IV is.

If the first i subblocks of the plaintext are unchanged, the corresponding ciphertext subblocks are unchanged, too. Further, the XOR of the old and new version of the first changed plaintext subblock equals the XOR of the corresponding ciphertext subblocks. This provides a characteristic pattern to identify the disk sector where the modification has taken place.

If a shrewd attacker A talks a naive user U into certain modifications of SFS-protected data, A can verify whether the modifications have taken place. A can even locate where these data are stored on disk. E.g. if A writes a patch as software "bug fix", A can easily check whether U actually patched the software.

We conclude that there is a weakness in SFS's encryption function which allows an enemy to launch a *chosen plaintext attack*. Nevertheless, SFS users need not panic—only under special circumstances the threat does become practical. The cipher can be repaired by methods described in this paper.

10 Example Ciphers

The original Luby-Rackoff cipher is represented by framework (5) with $b = 2$. How efficient is its implementation based on a dedicated hash function h?

Each round, h is evaluated once with an l-bit input. When encrypting two subblocks, we hash three of them. One naively might argue that the cipher's throughput is $\frac{2}{3}$ of the hash function's throughput—if we ignore the small effort for the additional XORs. This is true if the hash function's throughput is constant for different input sizes. Dedicated hash functions are typically optimized to hash large inputs at high speed—but may be slow on small inputs. This is what the speed-up effect depends on.

In the following, we present two block ciphers with moderately large blocks (i.e. 640 bits), one based on a MD5, the other one on RIPE-MD. We also consider their variants based on "large blocks".

Both MD5 and RIPE-MD compute a 128 bit hash. At their start, 128 chaining bits are initialized with a fixed value. Given the chaining bits and 512 message bits, the internal compression function outputs 128 bits. These serve as the new chaining bits or—when there are no more message bits—as the final hash value.

Before being fed into the internal compression function, the message is padded so that its length is a multiple of 512. The last bits of the padded message contain information about the message's length. If the message length is a multiple of 512 bits or just below, one additional block is needed for the length information. While padding increases the security of a hash function which operates on variable-size inputs, it is useless for us.

Hashing as few as 128 bits invokes one evaluation of MD5's resp. RIPE-MD's internal compression function—hashing $512x$ bits invokes about x evaluations. Hence, encrypting with Luby-Rackoff is about six times slower than hashing.

Other MD4-like hash functions like SHA and HAVAL work the same way, except for the exact numbers of chaining, message, and output bits.

We denote the internal compression function by $H : \{0,1\}^{512} \longrightarrow \{0,1\}^{128}$. Let us ignore padding. Then our cipher invokes $\lceil (b-1)/4 \rceil$ calls of H in the first round, $b-1$ in the second, and 1 call in the third round. This is a total of $b + \lceil (b-1)/4 \rceil$, hence the throughput is proportional to

$$\frac{b}{b + \lceil \frac{b-1}{4} \rceil}.$$

For $b = 5$, we can use H as random function and achieve a throughput proportional to $4/5$. For "large b", padding does not significantly slow down hashing—by using MD5 resp. RIPE-MD itself as random function we also achieve a throughput proportional to $4/5$. While operating on larger blocks and thus seemingly "doing more work", our cipher actually is about 20% faster than its Luby-Rackoff counterpart. For our example cipher, we deal with $b = 5$, i.e. fix the block size to 640 bits:

- Use $H : \{0,1\}^{512} \longrightarrow \{0,1\}^{128}$ as described above.
- Use K_1, $K_2 = (K_{2,1}, \ldots, K_{2,4})$, and K_3 as 512 bit round keys. Generate the round keys pseudorandomly from a master key $K \in \{0,1\}^*$ by
  $(K_1, K_2, K_3) = \big(\mathrm{MD5}(K), \ldots, \mathrm{MD5}(K+11)\big) \in \{0,1\}^{3*512}$ or
  $(K_1, K_2, K_3) = \big(\text{RIPE-MD}(K), \ldots, \text{RIPE-MD}(K+11)\big) \in \{0,1\}^{3*512}$.
- Divide the plaintext into 5 subblocks L, $R_{(1)}$, $R_{(2)}$, $R_{(3)}$, and $R_{(4)}$.
- For encryption and decryption use

$$\begin{aligned} S &= L \oplus H\big(K_1 \oplus (R_{(1)}, R_{(2)}, R_{(3)}, R_{(4)})\big), \\ T_{(i)} &= R_{(i)} \oplus H\big(K_{2,i} \oplus S, K_{2,i}, K_{2,i}, K_{2,i}\big) \quad (1 \le i \le 4), \\ U &= S \oplus H\big(K_3 \oplus (T_{(1)}, T_{(2)}, T_{(3)}, T_{(4)})\big). \end{aligned} \tag{7}$$

– The ciphertext is the concatenation of U, $T_{(1)}$, $T_{(2)}$, $T_{(3)}$, and $T_{(4)}$.

In contrast to the $b > 2$ case of (5), even the *decryption* function defined by equations (7) is pseudorandom. This feature is of minor practical relevance, but we get it for free since we need four subblocks as input for H anyway.

In order to encrypt 640 bits, i.e. five 128 bit subblocks, the function H is invoked 6 times. This is equivalent to hashing $6 * 512$ bits, so our example ciphers are about $6 * 512/640 = 4.8$ times slower than the underlying hash functions. In principle, this also is true for the "large block" variants based on framework (5)—but look out for cache misses!

Due to Roe's [16] results, we expect the MD5 based example cipher to be as fast as Blowfish, while its RIPE-MD counterpart seems to be 20% slower.

11 Concluding Remarks

The ciphers described here are provably secure against *adaptive chosen plaintext attacks* if the underlying primitive is secure. Resistance against such attacks is commonly accepted as a sufficient security criterion for secret-key block ciphers.

Note that neither Luby's and Rackoff's three-round block cipher nor the ciphers represented by (5) or (7) are secure against *combined adaptive chosen plaintext/chosen ciphertext attacks*. What, if we aim for security even against this stronger type of attack?

It is straightforward to append a fourth round to the cipher from figure 1. Luby and Rackoff showed that the resulting cipher, $\psi(f_1, f_2, f_3, f_4)$, is secure even against those attacks. It requires two round functions $f_1, f_3 : \{0,1\}^r \longrightarrow \{0,1\}^l$ and two round functions $f_2, f_4 : \{0,1\}^l \longrightarrow \{0,1\}^r$. But now, by choosing $l \neq r$ we achieve no speed-up. E.g. such a cipher, based on RIPE-MD or MD5, has to be eight times slower than the underlying hash function.

If we allow the ciphertext block to be slightly longer than the plaintext block, it is easy to fix the problem without loosing encryption speed:

– Use $h : \{0,1\}^* \longrightarrow \{0,1\}^l$ as above and $K_1, K_2 = (K_{2,1}, \ldots, K_{2,b}) \in \{0,1\}^{bl}$ as round keys—maybe generated pseudorandomly from a smaller master key.
– In order to encrypt the bl bit plaintext $R = (R_{(1)}, \ldots, R_{(b)})$, first compute $S := h(R \oplus K_1)$, then $T_{(i)} := R_{(i)} \oplus h(S \oplus K_{2,i})$.
– The $(b+1)l$ bit ciphertext is $(S, T_{(1)}, \ldots, T_{(b)})$.
– Decrypt by $R_{(i)} := T_{(i)} \oplus h(S \oplus K_{2,i})$, but output R only if $S = h(R \oplus K_1)$.

This cipher is secure with respect to adaptive chosen plaintext attacks for similar reasons as framework (5). And, thanks to S, the cipher cannot be used as a chosen ciphertext oracle. It is infeasible to find two different plaintexts R and R' with $h(R \oplus K_1) = h(R' \oplus K_1)$. Also, given R_1, R_2, …, and $S_1 = h(R_1 \oplus K_1)$, $S_2 = h(R_2 \oplus K_1)$, …, it is infeasible to find a pair (R^*, S^*) with $R^* \neq R_1$, $R^* \neq R_2$, … and $S^* = h(R^* \oplus K_1)$—if K_1 is secret and random.

Some hash functions are based on a block cipher like internal mapping and hence are easy to convert into block ciphers. A similar idea used Kaliski and Robshaw [8]. Disadvantages of this approach are:

- Not all hash functions have this property.
- For decryption, we must deal with some of the compression function's details. The compression function can't be treated like a black box—we lose the ease of implementation and the memorizability of Luby-Rackoff like ciphers.
- If the blocksize superseeds the number of chaining bits, as in the Kaliski-Robshaw proposal, it seems difficult still "*to relate the security of the block cipher to the security of the hash function*" [8].

Acknowledgement
The comments of an anonymous referee helped a lot to improve this paper.

References

1. R. Anderson, *The Classification of Hash Functions*, in: Fourth IMA conference on cryptography and coding, 83–93.
2. R. Anderson, *On Fibonacci Keystream Generators*, in: Fast Software Encryption (1994), Springer LNCS 1008, 346–352.
3. M. Blaze, B. Schneier, *The MacGuffin Block Cipher Algorithm*, in: Fast Software Encryption (1994), Springer LNCS 1008, 96–110.
4. O. Goldreich, S. Goldwasser, S. Micali, *How to Construct Random Functions*, in: 21st ACM SToC, 25–32, 1989.
5. P. Gutmann, documentation to SFS release 1.20, especially parts `SFS6.DOC` and `SFS7.DOC`, URL: "`http://www.cs.auckland.ac.nz/pgut01/sfs.html`", 1995.
6. J. Håstad, *Pseudo-Random Generators under Uniform Assumptions*, in: 22st ACM SToC, 395–404, 1990.
7. R. Impagliazzo, L. Levin, M. Luby, *Pseudo-random generation from one-way functions*, in: 21st ACM SToC, 12–24, 1989.
8. B. Kaliski, M. Robshaw, *Fast Block Cipher Proposal*, in: Fast Software Encryption (1993), Springer LNCS 809, 1994, 33–40.
9. M. Luby, C. Rackoff, *How to construct pseudorandom permutations from pseudorandom functions*, in: SIAM J. Computing, Vol. 17, No. 2, 373–386.
10. U. Maurer, *A Simplified and Generalized Treatment of Luby-Rackoff Pseudorandom Permutation Generators*, in: EuroCrypt '92, 239–255.
11. R. Needham, D. Wheeler, *Tea, a Tiny Encryption Algorithm*, in: Fast Software Encryption (1994), Springer LNCS 1008, 363–366.
12. J. Pieprzyk, B. Sadeghiyan, *Design of Hashing Algorithms*, Springer, 1993
13. M. Rabin, *Fingerprinting by Random Polynomials*, Harvard Univ. Tech. Rep. TR-15-81, 1981.
14. *Race Integrity Primitives Evaluation (RIPE): final report*, RACE 1040, 1993.
15. P. Rogaway, D. Coppersmith, *A Software-Optimized Encryption Algorithm*, in: Fast Software Encryption (1993), Springer LNCS 809, 1994, 56–63.
16. M. Roe, *Performance of Block Ciphers and Hash Functions — One Year Later*, in: Fast Software Encryption (1994), Springer LNCS 1008, 359–362.
17. B. Schneier, *Applied Cryptography*, Wiley, 1993.
18. D. Wheeler, *A Bulk Data Encryption Algorithm*, in: Fast Software Encryption (1993), Springer LNCS 809, 1994, 127–134.

This paper was published at:
Fast Software Encryption (1996),
Springer LNCS 1039, 189–203.