On the Security of Remotely Keyed Encryption

Stefan Lucks

Institut für Numerische und Angewandte Mathematik
Georg–August–Universität Göttingen
Lotzestr. 16–18, D–37083 Göttingen, Germany
(email: lucks@math.uni-goettingen.de)

Abstract. The purpose of remotely keyed encryption is to efficiently realize a secret-key block cipher by sharing the computational burden between a fast untrusted device and a slow device trusted with the key. This paper deals with how to define the security of remotely keyed encryption schemes. Since the attacker can take over the slow device and actually take part in the encryption process, common definitions of the security of block ciphers have to be reconsidered.
Using random mappings, collision resistant hash functions, and stream ciphers as building blocks, the Random Mapping based Remotely Keyed (RaMaRK) encryption scheme is proposed. Also GRIFFIN is proposed, a fast new block cipher for flexible but large blocks. The RaMaRK scheme and GRIFFIN are provably secure if the underlying building blocks are secure.

1 Introduction

At the Fast Software Encryption conference 1996 in Cambridge, Blaze [5] proposed a new paradigm for secret-key block ciphers: *Remotely keyed encryption*. This means to share the workload for en- and decryption between a fast host and a slow card. The host is trusted with plaintexts and ciphertexts, but only the (hopefully tamper-resistant) card does know the key.

While Blaze's "remotely keyed encryption protocol" is a fresh and interesting approach to solve the paradoxical problem how to realize "high-bandwidth encryption with low-bandwidth smartcards", it also has some drawbacks. In Section 3.1 of his paper, Blaze himself mentions some security problems, but assumes "neither of these attacks is likely to pose a serious threat to most practical applications." In spite of this assumption, the current author believes that a block cipher realized by a remotely keyed encryption scheme should be as secure as usually demanded from other block ciphers.

One goal of this paper is to generalize the common notions of block cipher security. For block ciphers, one usually considers attacks where the attacker can choose ciphertexts and decrypt them, and/or choose plaintexts and encrypt them. For remotely keyed encryption, we have to consider the case when the attacker does take part in the encryption or decryption protocol as the host. A second goal is to come up with a new remotely keyed encryption scheme. This has to be provably secure with respect to the generalized notions of security if its building blocks are secure. As a side result, the block cipher GRIFFIN realized by our scheme is a fast and secure block cipher for large blocks—and of practical interest even outside the scope of remotely keyed encryption, see section 6.

Throughout this paper, "random" always means "according to the uniform probability distribution". If x is randomly chosen from the set S, we write $x \in_{\mathrm{R}} S$. If x and y are independent random values from S, then $(x, y) \in_{\mathrm{R}} S^2$. By $x \oplus y$ we denote the bit-wise XOR of x and y.

2 How to Attack the RKEP

In this section, we describe Blaze's *remotely keyed encryption protocol* (RKEP) [5] and point out some of its weaknesses.

For the RKEP, we need a collision resistant hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^b$ and a secret-key block cipher with b-bit blocks. Given the key $K \in \{0,1\}^k$, the encryption function is $E_K : \{0,1\}^b \longrightarrow \{0,1\}^b$ and the decryption function is D_K. To cover the case when $k \neq b$, one needs a public mapping $M : \{0,1\}^b \longrightarrow \{0,1\}^k$. By $P = (P_1, \ldots, P_n) \in \{0,1\}^{nb}$ we denote a plaintext block, which can be divided into n subblocks $P_1, \ldots, P_n$ each of b bits. $C = (C_1, \ldots, C_n) \in \{0,1\}^{nb}$ is the corresponding ciphertext block. When encrypting P or decrypting C, an intermediate block $I = (I_1, \ldots, I_n) \in \{0,1\}^{nb}$ is considered.

Encrypting with the RKEP works as follows:

1. Given P, the host computes I by $I_i := P_i \oplus H(P_1)$ for $i \in \{2, \ldots, n\}$ and then $I_1 := P_1 \oplus H(I_2, \ldots, I_n)$. We abridge this to $I := \text{Modify}(P)$.
2. I_1 is sent to the card, which knows the secret key K and computes $C_1 := E_K(I_1)$ and $K_P := M(E_K(C_1))$. The pair (C_1, K_P) is sent back to the host. For the sake of shortness, we write $(C_1, K_P) := \text{Local}_K(I_1)$.
3. The host computes the remaining $n-1$ subblocks $C_2 := E_{K_P}(I_2 \oplus C_1)$, $C_3 := E_{K_P}(I_3 \oplus C_2)$, $\ldots$, $C_n := E_{K_P}(I_n \oplus C_{n-1})$ of the ciphertext C.

Decryption is similar:

1. Given the ciphertext C, the host sends C_1 to the card.
2. The card computes $I_1 := D_K(C_1)$, $K_P := M(E_K(C_1))$ and sends both values to the host. For short, we write $(I_1, K_P) := \text{Local}_K^{-1}(C_1)$.
3. The host computes $I_2 := D_{K_P}(C_2) \oplus C_1$, $I_3 := D_{K_P}(C_3) \oplus C_2$, $\ldots$, $I_n := D_{K_P}(C_n) \oplus C_{n-1}$.
   Also, the host computes the plaintext P from I: $P_1 := I_1 \oplus H(I_2, \ldots, I_n)$ and $P_i := I_i \oplus H(P_1)$ for $i \in \{2, \ldots, n\}$—in short: $P := \text{Modify}^{-1}(I)$.

Apart from enabling remotely keyed encryption, the RKEP can be seen as a block cipher with large (i.e. bn bit) blocks. Thus we can write $C := \text{RKEP}_K(P)$ for encrypting with the secret key K, and $P := \text{RKEP}_K^{-1}(C)$ for decrypting. For block ciphers, one usually requires: *If the attacker encrypts or decrypts at most q times, but has no further knowledge about the key K or plaintext-ciphertext pairs, there should be no more than q such pairs $(P^{(i)}, C^{(i)})$ with $C^{(i)} = RKEP_K(P^{(i)})$*

known to the attacker. As we will see below, the RKEP block cipher does not meet this requirement.

Now, we consider two attacks on the RKEP block cipher. The attacker, we call her Alice, has the unwitting help of Bob, who encrypts and decrypts for her. Attack I is a *chosen plaintext attack*, since given a plaintext P, Alice chooses two plaintexts $P' = (P'_1, \ldots P'_n)$ and $P'' = (P''_1, \ldots P''_n)$ different from P, receives the corresponding ciphertexts $C' = (C'_1, \ldots C'_n)$ and $C'' = (C''_1, \ldots C''_n)$ from Bob, and uses this knowledge to compute the ciphertext $C = \mathrm{RKEP}_K(P)$ without more help by Bob. Attack II is a *two-sided attack*, i.e. the attacker must be able to encrypt *and* to decrypt.

Attack I: Given a "suspicious" plaintext P Bob is not willing to encrypt for Alice, Alice chooses two plaintexts P' and P'' which appear random. If Bob gives her $C' = \mathrm{RKEP}_K(P')$ and $C'' = RKEP_K(P'')$, she can derive the pair (C_1, K_P) of values required to compute $C = \mathrm{RKEP}_K(P)$ from this, without further asking Bob. The attack requires the following six steps:

1. Alice computes $I = \mathrm{Modify}(P)$.
2. She chooses any $I' \neq I$, except for $I'_1 = I_1$.
3. She computes $P' = \mathrm{Modify}^{-1}(I')$ and asks Bob for $C' = \mathrm{RKEP}_K(P')$. (Note that $I'_1 = I_1$ and hence $C'_1 = C_1$.)
4. She chooses any $I'' \neq I$, except for $I''_1 = C'_1$.
5. She computes $P'' = \mathrm{Modify}^{-1}(I'')$ and asks Bob for $C'' = \mathrm{RKEP}_K(P'')$. (Now $C''_1 = E_K(I''_1) = E_K(C'_1) = E_K(C_1)$.)
6. She computes $K_P = M(C''_1)$.

Attack II: Given a ciphertext C Bob is not willing to decrypt for her, Alice chooses a ciphertext C' different from C and a plaintext P''. Similarly to the above attack, Alice can find the pair (C_1, K_P) she needs to compute $P = \mathrm{RKEP}_K(C)$, once she is given $P' = \mathrm{RKEP}_K^{-1}(C')$ and $C'' = RKEP_K(P'')$:

1. Alice chooses any $C' \neq C$, except for $C'_1 = C_1$.
2. She asks Bob for $P' = \mathrm{RKEP}_K^{-1}(C')$.
3. She computes $I' = \mathrm{Modify}(P')$. (Now she knows $I_1 = I'_1$.)
4. As in the last three steps of attack I, Alice computes $K_P = M(E_K(I_1))$.

For attack III, we will not consider attacks on the RKEP block cipher, but examine how to misuse a given smartcard. Here, Alice does not need Bob's help but can decrypt ciphertexts for herself, using a "decrypt only" smartcard. This is a reasonable demand e.g. for pay-TV applications, where (paying) customers are given a smartcard to decrypt broadcast data (i.e. the TV program). Customers should not be able to encrypt. Otherwise they could encrypt and broadcast video data for themselves, and thus forge the pay-TV program.

We consider a smartcard to compute the function Local_K^{-1}—but not Local_K. We assume that with significant probability it is possible to determine $E_K(C_1)$ from $M(E_K(C_1))$. This assumption appears to be reasonable, since Blaze only requires the public function M to map a b-bit string to a k-bit key string, but does not demand M to be either secret or a one-way function ([5], top of p. 35).

Attack III: Given a target plaintext P, Alice proceeds like this:

1. She computes $I := \text{Modify}(P)$.
2. She feeds I_1 into her smartcard to get
$$\big(\text{dummy}, X\big) = \text{Local}_K^{-1}(I_1) = \big(\ldots, M(E_K(I_1))\big).$$
3. She determines C_1 with $X = M(C_1)$.
4. She feeds C_1 into the smartcard to get
$$\big(Y, K_P\big) = \text{Local}_K^{-1}(C_1) = \big(D_K(C_1), M(E_K(C_1))\big).$$

Note that if $b > k$, exaustive search over the 2^{b-k} possible values C_1' with $M(C_1') = X$ is needed to find the unique C_1 with $I_1 = Y = D_k(C_1)$. Once Alice knows $(C_1, K_P) = \text{Local}_K(I_1)$, computing $C_2, \ldots, C_n$ is easy for her.

We will not discuss under which circumstances the above attacks become practical. Possibly, our attacks are of no relevance for the applications Blaze considered for the RKEP. However, the RKEP block cipher clearly has some properties, a *good* block cipher should not have.

3 When is Remotely Keyed Encryption "Secure"?

A *remotely keyed encryption scheme* is a protocol to distribute the computational burden for a B-bit block cipher between two parties, a *host* and a *card*. (The "card" could either be a smartcard connected to the host, or something quite different, e.g. an "encryption server" in a computer network.) The host knows plaintext and ciphertext, but only the card is trusted with the key.[1] The protocol is divided into two subprotocols, an *encryption protocol* and a *decryption protocol*.

Given a B-bit input, either to encrypt or to decrypt, such a subprotocol runs like this: The host sends a *challenge value* to the card, depending on the input, and the card replies a *response value*, depending on both the challenge value and the key. E.g. in the case of the RKEP encryption protocol, I_1 is the challenge value, and the pair (C_1, K_P) is the response value, while for RKEP decryption C_1 is the challenge value, and (I_1, K_P) is the response value. Exchanging challenge and response values can be iterated. During one run of a subprotocol every challenge value depends on the input and the previously given response values and the response values depend on the key and the previous challenge values. We may assume that neither the overall number of bits for the challenge values, nor the overall number of bits for the response values exceed β, where $\beta \ll B$.

For a key $K \in \{0,1\}^k$, the encryption protocol realizes the encryption function $\texttt{Encrypt}_K : \{0,1\}^B \longrightarrow \{0,1\}^B$ and the decryption protocol the decryption function $\texttt{Decrypt}_K : \{0,1\}^B \longrightarrow \{0,1\}^B$, such that for every plaintext $X \in \{0,1\}^B$ the equation $X = \texttt{Decrypt}_K(\texttt{Encrypt}_K(X))$ holds.[2]

[1] Such a scheme can also be seen as the card's "mode of operation".

[2] For $B^* \geq B$ this definition can be generalized to $\texttt{Encrypt}_K : \{0,1\}^B \longrightarrow \{0,1\}^{B^*}$ and $\texttt{Decrypt}_K : \{0,1\}^{B^*} \longrightarrow \{0,1\}^B$, as long as $X = \texttt{Decrypt}_K(\texttt{Encrypt}_K(X))$ is valid for all plaintexts $X \in \{0,1\}^B$. But for the purposes of this paper, we don't need that generalization.

In order to define the security of such a protocol, we first need to describe the security of block ciphers. In this paper, we only consider block ciphers with ciphertexts of the same size as the plaintexts. Encrypting with such a block cipher means to apply a key-dependent permutation g to the plaintext, decrypting to apply its inverse g^{-1} to the ciphertext.

In cryptography, one often considers a *chosen plaintext attack*, where attackers can encrypt plaintexts of their choice. Here, we consider an even stronger type of attack, the *two-sided attack*, often called "combined adaptive chosen plaintext/chosen ciphertext attack", where attackers are able to encrypt plaintexts of their choice and decrypt ciphertexts of their choice.

A block cipher is *forgery secure*, if after q encryptions resp. decryptions of chosen inputs, the attacker can know at most q plaintext-ciphertext pairs $(P^{(1)}, C^{(1)})$, ..., $(P^{(q)}, C^{(q)})$, but no $(P, C) \notin \{(P^{(1)}, C^{(1)}), \ldots, (P^{(q)}, C^{(q)})\}$, with $C = \texttt{Encrypt}_K(P)$ (without encrypting or decrypting again).

In the case of remotely keyed encryption, we need to consider attacks, where instead of encrypting plaintexts or decrypting ciphertexts the attackers act as the host when executing the encryption resp. decryption protocol.

Definition: A remotely keyed encryption scheme is *forgery secure*, if after q executions of the encryption resp. decryption protocol with arbitrarily chosen challenge values, the attacker can know no more than q plaintext-ciphertext pairs $(P^{(1)}, C^{(1)})$, ..., $(P^{(q)}, C^{(q)})$ which are *valid*, i.e. $C_i = \texttt{Encrypt}_K(P_i)$.

Another property of block ciphers is, when attackers can operate only in one direction, e.g. choose plaintexts and encrypt, they should be unable to solve the problem to compute in the reverse direction, i.e. to decrypt a given ciphertext.

Definition: A remotely keyed encryption scheme is *inversion secure*,

- if for attackers able to execute the encryption protocol it is infeasible to decrypt a randomly chosen ciphertext, and
- if for attackers able to execute the decryption protocol it is infeasible to encrypt a randomly chosen plaintext.

Often, the security of cryptographic primitives is defined as the non-existence of statistical tests to detect non-randomness properties. This point of view leads to an even stronger definition of the security of block ciphers than forgery-security: A B-bit block cipher is called *pseudorandom*, if for two-sided attackers it is infeasible to distinguish whether they are given a random permutation $p : \{0,1\}^B \longrightarrow \{0,1\}^B$ and its inverse, or the encryption function $\texttt{Encrypt}_K : \{0,1\}^B \longrightarrow \{0,1\}^B$ and the decryption Function $\texttt{Decrypt}_K$ depending on the secret key K. Attackers who participate in an encryption or decryption protocol learn the card's response values and thus know that the encryption is not a random permutation. Hence when considering pseudorandomness, we can't consider attackers which execute the encryption or decryption protocol.

Definition: A remotely keyed encryption scheme is *pseudorandom*, if the block cipher it realizes is pseudorandom.

Note that the RKEP is neither forgery secure (cf. attacks I and II), nor inversion secure (cf. attack III), nor pseudorandom (cf. attacks I and II). Even if we only consider chosen plaintext attacks, the RKEP block cipher is neither forgery secure nor pseudorandom (cf. attack I).

4 The RaMaRK Encryption scheme

Now we describe the *Random Mapping based Remotely Keyed (RaMaRK) Encryption scheme*. The name comes from one of our building blocks, a *fixed size random mapping* $f : \{0,1\}^b \longrightarrow \{0,1\}^b$. For the proofs of security, we assume f to be a random function (a "random oracle" in the sense of Bellare and Rogaway [4]), i.e. for $s \neq t$: $(f(s), f(t)) \in_{\mathrm{R}} \{0,1\}^{2b}$.

Except when b is tiny, one can't actually implement such random functions—one would have to store $b2^b$ bits. In practice, one assumes f to be a pseudorandom function depending on a small secret key and undistinguishable from a random function for everyone without knowledge of the key. This could be done e.g. by using a block cipher or a dedicated hash function. Note that realizing pseudorandom mappings from dedicated hash functions must be done with great care to be secure (cf. [9]). Also note that b must be large enough—performing close to $2^{b/2}$ encryptions has to be infeasible.[3] We use three building blocks:

1. Random mappings $f_i : \{0,1\}^b \longrightarrow \{0,1\}^b$, as described above.
2. A hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^b$. H has to be *collision resistant*, i.e. it has to be infeasible to find any $t, u \in \{0,1\}^*$ with $u \neq t$ but $H(u) = H(t)$.
3. A pseudorandom bit generator (i.e. a "stream cipher") $S : \{0,1\}^b \longrightarrow \{0,1\}^*$. We restrict ourselves to $S : \{0,1\}^b \longrightarrow \{0,1\}^{B-2b}$. If the seed $s \in \{0,1\}^b$ is randomly chosen, the bits produced by $S(s)$ have to be undistinguishable from randomly generated bits.
   For theorem 3 we need an *additional property*: If s is secret and attackers choose $t_1, t_2, \ldots \in \{0,1\}^b$ with $t_i \neq t_j$ for $i \neq j$ and receive outputs $S(s \oplus t_1), S(s \oplus t_2), \ldots$, it has to be infeasible for the attackers to distinguish these outputs from independently generated random bit strings of the same size. Hence, such a construction behaves like a random mapping $\{0,1\}^b \longrightarrow \{0,1\}^{B-2b}$, but actually is a pseudorandom one, depending on the secret s.

Based on these building blocks, we realize a remotely keyed encryption scheme to encrypt blocks of any size $B \geq 3b$, see figure 1. In contrast to Blaze's RKEP, B need not be a multiple of b.

We represent the plaintext by (P, Q, R) and the ciphertext by (A, B, C), where $(P, Q, R), (A, B, C) \in \{0,1\}^b \times \{0,1\}^b \times \{0,1\}^{B-2b}$. For the protocol de-

[3] If we only are given a random mapping $f' : \{0,1\}^{b'} \longrightarrow \{0,1\}^{b'}$ with b' just large enough that performing close to $2^{b'}$ encryptions is infeasible, we can apply Aiello's and Venkatesan's work [1] to construct a random mapping $f : \{0,1\}^b \longrightarrow \{0,1\}^b$ with $b = 2b'$. Provably $\Omega(2^{b/2})$ queries are needed to distinguish with $\Theta(1)$ probability between f and a truly random b-bit to b-bit function.

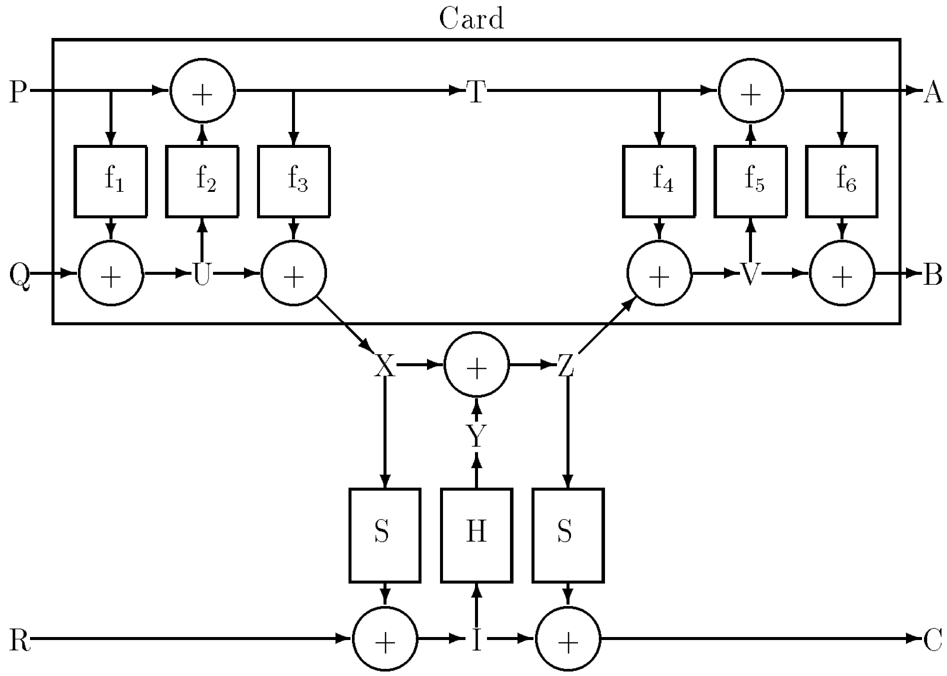

Fig. 1. The RaMaRK encryption protocol.

scription we also consider intermediate values $T, U, V, X, Y, Z \in \{0,1\}^b$, and $I \in \{0,1\}^{B-2b}$. The encryption protocol works as follows:

1. Given the plaintext (P, Q, R), the host sends P and Q to the card.
2. The card computes $U = f_1(P) \oplus Q$, $T = f_2(U) \oplus P$, and sends $X = f_3(T) \oplus U$ to the host.
3. The host computes $I = S(X) \oplus R$, $Y = H(I)$, sends $Z = X \oplus Y$ to the card, and computes $C = S(Z) \oplus I$.
4. The card computes $V = f_4(T) \oplus Z$ and sends the two values $A = f_5(V) \oplus T$ and $B = f_6(A) \oplus V$ to the host.

Decrypting (A, B, C) is done like this:

1. The host sends A and B to the card.
2. The card computes $V = f_6(A) \oplus B$, $T = f_5(V) \oplus A$, and sends $Z = f_4(T) \oplus V$ to the host.
3. The host computes $I = S(Z) \oplus C$, $Y = H(I)$, sends $X = Z \oplus Y$ to the card, and computes $R = S(X) \oplus I$.
4. The card computes $U = f_3(T) \oplus X$ and sends the two values $P = f_2(U) \oplus T$ and $Q = f_1(P) \oplus U$ to the host.

One can easily verify that by first encrypting any plaintext using any key, then decrypting the result using the same key, one gets the same plaintext again.

5 The Security of the RaMaRK encryption scheme

Next, we prove the forgery security, the inversion security, and the pseudorandomness of the RaMaRK scheme.

Theorem 1. *A two-sided attack against the RaMaRK encryption scheme to find q valid plaintext-ciphertext pairs by $q-1$ protocol executions succeeds with at most the probability $q^2 2/2^b$.*

Proof. By P_i, Q_i, X_i, Z_i, A_i, and B_i we denote the challenge and response values of the ith protocol execution—independently of whether the encryption protocol is executed, i.e. (P_i, Q_i) and Z_i are challenge and X_i and (A_i, B_i) response values, or the decryption protocol is executed and the role of challenge and response values is reversed. Similarly are the subscripts for I_i, Y_i, T_i, U_i, and V_i used.

Since H is collision resistant, for every Y_i at most one I_i with $H(I_i) = Y_i$ can be known to the attacker and thus every triple (P_i, Q_i, Y_i) for the encryption protocol corresponds to at most one plaintext (P_i, Q_i, R_i) known to the attacker, and every triple (A_i, B_i, Y_i) corresponds to at most one ciphertext. It doesn't help attackers to repeat the challenge values of a previous protocol execution. For $i \neq j$ we can assume $(A_i, B_i, C_i) \neq (A_j, B_j, C_j)$ and $(P_i, Q_i, Z_i) \neq (P_j, Q_j, Z_j)$. After every protocol execution, the attacker can choose whether next to execute the encryption or the decryption protocol.

At first, we consider the **encryption protocol**. The attacker sends P_r and Q_r to the card. For every $i \in \{1, \ldots, r-1\}$ we need to distinguish whether $P_i = P_r$ and $Q_i = Q_r$ or not.

Case I: $P_i = P_r$ and $Q_i = Q_r$. In this case, the corresponding internal values are equal, i.e. $T_i = T_r$ and $X_i = X_r$, but $Z_i \neq Z_r$ and thus $V_i \neq V_r$, $(A_i, A_r) \in_{\mathrm{R}} \{0,1\}^{2b}$, and $\mathrm{prob}[A_i = A_r] \leq 1/2^b$. If $A_i \neq A_r$ then $(B_i, B_r) \in_{\mathrm{R}} \{0,1\}^{2b}$. Thus with a probability $\geq 1 - 1/2^b$, the ciphertext fractions (A_i, B_i) and (A_r, B_r) are independently chosen random values.

Case II: $P_i \neq P_r$ or $Q_i \neq Q_r$. If $P_i = P_r$ and $Q_i \neq Q_r$, then $U_i \neq U_r$. If $P_i \neq P_r$, then $f_1(P_i)$ and $f_1(P_r)$ are two independent random values, we write $(f_1(P_i), f_1(P_r)) \in_{\mathrm{R}} \{0,1\}^{2b}$, and so are U_i and U_r, thus $\mathrm{prob}[U_i = U_r] \leq 1/2^b$.
If $U_i \neq U_r$, then $(T_i, T_r) \in_{\mathrm{R}} \{0,1\}^{2b}$, hence $\mathrm{prob}[T_i = T_r] \leq 2/2^b$.
If $T_i \neq T_r$, then $(V_i, V_r) \in_{\mathrm{R}} \{0,1\}^{2b}$, $\mathrm{prob}[V_i = V_r] \leq 3/2^b$, $\mathrm{prob}[(A_i, A_r) \in_{\mathrm{R}} \{0,1\}^{2b}] \geq 1 - 3/2^b$, and $\mathrm{prob}[A_i = A_r] \geq 1 - 4/2^b$. Thus with a probability not below $1 - 4/2^b$, the ciphertext fractions (A_i, B_i) and (A_r, B_r) are independently chosen random values.

Due to the symmetric construction of the RaMaRK scheme, similar arguments apply as well if the attacker runs the **decryption protocol**.

Finding q valid plaintext-ciphertext pairs with $q-1$ protocol executions essentially means to find a plaintext (P_q, Q_q, R_q) where one can predict the corresponding ciphertext (A_q, B_q, C_q) without having used either $((P_q, Q_q), Z_q)$ or $((A_q, B_q), X_q)$ as challenge values for the encryption resp. decryption protocol.

We consider the second response value $w_i \in \{0,1\}^{2b}$ the attacker receives during the ith protocol execution, i.e. (A_i, B_i) when encrypting and (P_i, Q_i) when decrypting. Note that after $q-1$ protocol executions, the attacker needs to predict w_q. Random response values can't be predicted, the attacker can do no better than to deliberately guess one. There are $q(q-1)/2$ sets $\{i,j\}$ with $i \neq j$ and $i, j \in \{1, \dots, q\}$, hence the probability that after $q-1$ protocol executions the attacker can either predict or guess w_q is at most $q^2 2/2^b$. □

Theorem 1, gives us an exact upper bound for the probability of success of forgery attacks against our scheme, depending on the number $q-1$ of queries. For the sake of simplicity and shortness, in the following we omit such an exact treatment and assume that there are only $q = o(2^{b/2})$ queries. Note that the constants hidden by the asymptotics are quite small and can be found similarly to the proof of theorem 1.

Theorem 2. *The RaMaRK scheme is secure against inversion attacks.*

Sketch of proof: We restrict ourselves attackers who only can execute the encryption protocol. For reasons of symmetry, the proof for attackers only able to execute the decryption protocol is the same.

We consider the permutation $g : \{0,1\}^{2b} \longrightarrow \{0,1\}^{2b}$ with $g(P,Q) = (T,X)$. Given the ciphertext (T,X), our problem is to find a plaintext (M,N) with $g(M,N) = (T,X)$, where we are allowed to evaluate g, but not its inverse g^{-1}. We choose random functions f_4, f_5, and f_6, the value $R \in_{\mathrm{R}} \{0,1\}^{B-2b}$, and compute the corresponding ciphertext (A,B,C). Now we can simulate the card's part of the encryption protocol. If one could find the plaintext (P,Q,R) corresponding to (A,B,C), one also could break g. But since $q = o(2^{b/2})$ and g is a Luby-Rackoff cipher, chosen plaintext attacks on it are infeasible [6]. □

Theorem 3. *The RaMaRK scheme is pseudorandom.*

Sketch of proof: Reconsider the proof of theorem 1. At the rth point of time, the attacker either chooses a plaintext (P_r, Q_r, R_r) for encryption, or a ciphertext (A_r, B_r, C_r) for decryption.

First, we consider the case when a plaintext is chosen and encrypted. For every $i \in \{0, \dots, r-1\}$, we need to distinguish whether $P_i = P_r$ and $Q_i = Q_r$, or not.

Case I: If $P_i = P_r$ and $Q_i = Q_r$ then $R_i \neq R_r$ and thus $I_i \neq I_r$, otherwise the attacker would not get any new information.

Case II: If $P_i \neq P_r$ or $Q_i \neq Q_r$ then $\mathrm{prob}[T_i \neq T_r] \leq 2/2^b$, and if $T_i \neq T_r$ then $(X_i, X_r) \in_{\mathrm{R}} \{0,1\}^{2b}$ thus $S(X_i)$ and $S(X_r)$ are undistinguishable from independent $(B-2b)$-bit strings chosen at random—and so are I_i and I_r.

For reasons of symmetry, the same arguments apply if a ciphertext is chosen and decrypted.

Now $q = o(2^{b/2})$, i.e. $q^2 \ll 2^b/2$, thus we can expect all I_1, I_2, ..., I_q to be pairwise distinct. Then all Y_1, Y_2, ..., Y_q are pairwise distinct, too—otherwise the collision resistant hash function H would be broken.

From the proof of theorem 1 we can deduce that a part of the output—(A, B) when encrypting and (P, Q) when decrypting—is undistinguishable from randomly chosen bit-strings with a probability not below $1-q^2 2/2^b$. The pseudorandomness of the remaining part can be deduced from the *additional property* of the stream cipher S. □

When designing cryptographic algorithms and protocols one often needs collision resistant hash functions with more properties [2]. For our proof of security, collision resistance of the hash function H is sufficient. But if H behaves like a random oracle [4], we can abandon the demand for the *additional property* of S.

6 The Block Cipher GRIFFIN

Even if remotely keyed encryption is not required, GRIFFIN, the block cipher realized by the RaMaRK scheme, is of some interest of its own.

In 1996, Anderson and Biham [3] proposed two block ciphers for flexible but large blocks, BEAR and LION. A similar proposal is the block cipher BEAST, see [8]. All three block ciphers depend on using a hash function H and a stream cipher S as building blocks, BEAST also needs a fixed size pseudorandom mapping.

With respect to chosen ciphertext attacks, BEAR, LION, and BEAST are pseudorandom (see [7] for a proof). But neither of the three ciphers is pseudorandom as defined in this paper, i.e. with respect to two-sided attacks. Luby's and Rackoff's attack on three-round Luby-Rackoff ciphers [6] works well for BEAR, LION, and BEAST. Therefore, Anderson and Biham [3] also proposed LIONESS, a block cipher pseudorandom even with respect to two-sided attacks. BEAR requires to evaluate H twice and S once, LION requires to evaluate S twice and H once, and LIONESS requires to evaluate H twice and S twice, where the inputs of H and the outputs of S are about as large as the cipher's blocks. As suggested by Anderson and Biham, we consider using the hash function SHA-1 for H and the stream cipher SEAL for S. SEAL appears to be faster than SHA-1, at least if one ignores SEAL's key set-up time. Thus, if the blocks are large, LION should be faster than BEAR, and LIONESS slower.

On a 133 MHz DEC Alpha machine (a "sandpiper") and for 1 MBit blocks, Anderson and Biham measured 13.62 MBit/sec for BEAR and 18.68 MBit/sec for LION. We expect LIONESS to operate at about 11.8 MBit/sec under the same conditions.

Due to theorem 3, GRIFFIN is pseudorandom with respect to two-sided attacks, too. For large blocks (e.g. $B = 1$ MBit), the effort to evaluate the fixed size pseudorandom mappings f_1, ..., f_6 is negligible. Then the speed of GRIFFIN is determined by evaluating S twice and H once. If we use SHA-1 and SEAL, and if the blocks are large, GRIFFIN can be expected to run at about the same speed as LION and thus significantly faster than its competitor LIONESS.[4]

References

1. W. Aiello, R. Venkatesan, "Foiling Birthday Attacks in Length-Doubling Transformations", in *Eurocrypt'96* (ed. U. Maurer), Springer LNCS 1070, 307–320, 1996.
2. R. Anderson, "The Classification of Hash Functions", in *Fourth IMA conference on cryptography and coding*, 83–93, 1993.
3. R. Anderson, E. Biham, "Two Practical and Provably Secure Block Ciphers: BEAR and LION", in *Fast Software Encryption* (ed. D. Gollmann), Springer LNCS 1039, 113–120, 1996.
4. M. Bellare, P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols", in *First ACM Conference on Computer and Communications Security*, ACM, 1993.
5. M. Blaze, "High-Bandwidth Encryption with Low-Bandwidth Smartcards", in *Fast Software Encryption* (ed. D. Gollmann), Springer LNCS 1039, 33–40, 1996.
6. M. Luby, C. Rackoff, "How to Construct Pseudorandom Permutations from Pseudorandom Functions", *SIAM J. Computing*, Vol. 17, No. 2, 373–386, 1988.
7. S. Lucks, "Faster Luby-Rackoff Ciphers", in *Fast Software Encryption* (ed. D. Gollmann), Springer LNCS 1039, 189–203, 1996.
8. S. Lucks, "BEAST: A Fast Block Cipher for Arbitrary Blocksizes", in *IFIP Conference on Communications and Multimedia Security* (ed. P. Horster), Chapman & Hall, 144–153, 1996.
9. B. Preneel, P. van Oorschot, "On the Security of Two MAC Algorithms", in *Eurocrypt '96* (ed. U. Maurer), Springer LNCS 1070, 19–32, 1996.

Fast Software Encryption (1997), Springer Verlag, to appear.

[4] For similar reasons, BEAST outperforms LION. Under the same conditions as above, we expect BEAST to run at about 23.6 MBit/sec [8].