USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

The following paper was originally published in the

USENIX Workshop on Smartcard Technology

Chicago, Illinois, USA, May 10–11, 1999

Remotely Keyed Encryption Using Non-Encrypting Smart Cards

Stefan Lucks and Rüdiger Weis
University of Mannheim

For more information about the USENIX Association:
Phone: 1 510 528 8649 FAX: 1 510 548 5738
Email: office@usenix.org WWW: http://www.usenix.org

Remotely Keyed Encryption Using Non-Encrypting Smart Cards

Stefan Lucks
Theoretische Informatik
Rüdiger Weis
Praktische Informatik IV
University of Mannhein
68131 Mannheim, Germany
lucks@th.informatik.uni-mannheim.de
rweis@pi4.informatik.uni-mannheim.de

Abstract

Remotely keyed encryption supports fast encryption on a slow smart card. For the scheme described here, even a smart card without a builtin encryption function, would do the job, e.g., a signature card.

1 Introduction

Many security relevant applications store secret keys on a tamper-resistant device, a *smart card*. Protecting the valuable keys is the card's main purpose. Typically, smart cards are slow. Using them for key-dependent operations such as en- and decrypting inherently must be slow as well, right? Wrong, paradoxically there is still a way of doing fast encryption using a slow card.

Often, smart cards are designed to support authentication or digital signatures instead of encryption. In this paper, we concentrate on the RaMaRK protocol, theoretically based on (pseudo)random mappings. Paradoxically enough: The RaMaRK protocol does not require the smart card itself to support encryption – support for hash functions, as built into many signature cards, is sufficient. In a world with lots of restrictions on the import, export or usage of encryption tools and much less restrictions regarding authentication or signature tools, this can be an important property.

2 Remotely Keyed Encryption

A *remotely keyed encryption scheme* (RKES) distributes the computational burden for a block cipher with large blocks between two parties, a *host* and a *card*. Figure 1 gives a general description of an RKES.

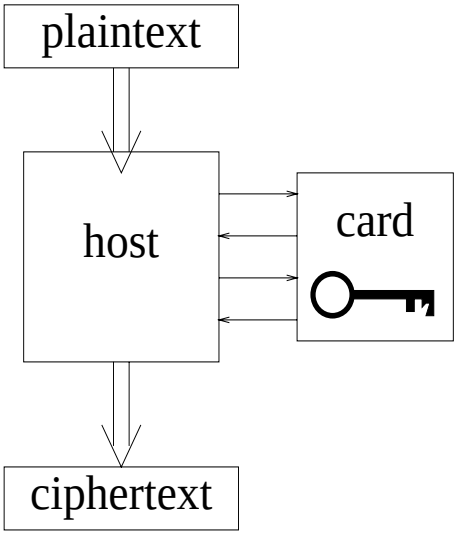

Figure 1: A generic RKES

We think of the host being a computer under the risk of being taken over by an adversary, while the

card can be a smart card, protecting the secret key. We do not consider attacks to break the tamper-resistance of the smart cards itself. The host knows plaintext and ciphertext, but only the card is trusted with the key.

An RKES consists of two protocols: the *encryption protocol* and the *decryption protocol*. Given a B-bit input, either to encrypt or to decrypt, such a protocol 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.

The notion of *remotely keyed encryption* is due to Blaze [2]. Lucks [8] pointed out some weaknesses of Blaze's scheme and gave formal requirements for the security of RKESs:

(i) *Forgery security*: If the adversary has controlled the host for $q-1$ interactions, she cannot produce q plaintext/ciphertext pairs.

(ii) *Inversion security*: An adversary with (legitimate) access to encryption must not be able do decrypt and vice versa.

(iii) *Pseudorandomness*: The encryption function should behave pseudorandomly for someone neither having access to the card, nor knowing the secret key.

While Requirements (i) and (ii) restrict the abilities of an adversary with access to the smart card, Requirement (iii) is only valid for *outsider adversaries*, having no access to the card. If an adversary could compute forgeries or run inversion attacks, she could easily distinguish the encryption function from a random one.

It is theoretically desirable, that a cryptographic primitive always appears to behave randomly for everyone without access to the key. So why not require pseudorandomness with respect to insider adversaries? In any RKES, the amount of communication between host and card should be smaller than the input length, otherwise the card could just do the complete encryption on its own. Since (at least) a part of the input is not handled by the smart card, and, for the same reasons, (at least) a part of the output is generated by the host, an insider adversary can easily decide that the output generated by herself is not random.

In 1998, Blaze, Feigenbaum, and Naor [3] found another way to define the pseudorandomness of RKESs. Their formal definition is quite complicated. Is is based on the adversary A gaining direct access to the card *for a certain amount of time*, making a fixed number of interactions with the card. When A has lost direct access to the card, the encryption function should appear to behave randomly, even for A. Recently, Lucks [9] described an "accelerated" RKES, which satisfies the security requirements of Blaze, Feigenbaum and Naor, but is significantly more efficient. Note that both schemes acutally require the card to execute encryption function, while this paper deals with remotely keyed encryption using non-encrypting smart cards.

Theoretically, one could define an encryption function based on random mappings and hence adapt the schemes of [3, 9] for the use of non-encrypting smartcards. Such a construction could be based on using Luby-Rackoff ciphers [6], or on one of the many refinements of them, such as the one in [7]. In practice, the resulting RKE-scheme would be quite inefficient, though.

3 RaMaRK Encryption scheme

In this section, we describe the *Random Mapping based Remotely Keyed (RaMaRK) Encryption scheme*, which uses several independent instances of a *fixed size random mapping* $f : \{0,1\}^b \longrightarrow \{0,1\}^b$. In practice, one uses pseudorandom functions[1] instead of truly random ones. The scheme is provably secure if its building blocks are, i.e., it satisfies requirements (i)–(iii) above, see [8]. Note that b must be large enough—performing close to $2^{b/2}$ encryptions has to be infeasible. We recommend to choose $b \geq 160$. By "$\oplus$" we denote the bit-wise XOR, though mathematically any group operation would do the job as well.

[1] If f is "pseudorandom", it is infeasible to distinguish between f and a truly random function - except if one knows the secret key.

We use three building blocks:

1. Key-dependent (pseudo-)random mappings $f_i : \{0,1\}^b \longrightarrow \{0,1\}^b$.

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 indistinguishable from randomly generated bits.

   In addition to pseudorandomness, the following property is needed: 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 $S(s \oplus t_1)$, $S(s \oplus t_2)$, ..., 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 $\{0,1\}^b \longrightarrow \{0,1\}^{B-2b}$, thought it 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 2.

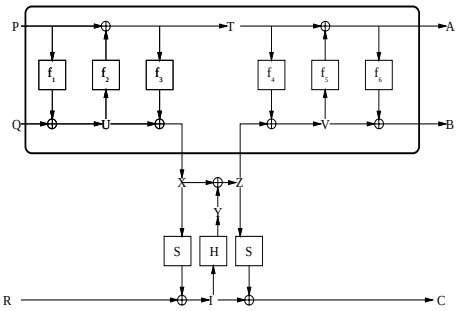

Figure 2: The RaMaRK encryption protocol

3.1 RaMaRK Encryption Protocol

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 description 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

   $$U = f_1(P) \oplus Q \text{ and } T = f_2(U) \oplus P,$$

   and sends

   $$X = f_3(T) \oplus U$$

   to the host.

3. The host computes

   $$I = S(X) \oplus R \text{ and } Y = H(I),$$

   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

   $$A = f_5(V) \oplus T \text{ and } B = f_6(A) \oplus V$$

   to the host.

3.2 RaMaRK Decryption Protocol

In order to decrypt the ciphertext (A, B, C), we need the following protocol:

1. Given the plaintext (A, B, C), the host sends A and B to the card.

2. The card computes

$$V = f_6(A) \oplus B \text{ and } T = f_5(V) \oplus A,$$

and sends

$$Z = f_4(T) \oplus V$$

to the host.

3. The host computes

$$I = S(Z) \oplus C \text{ and } Y = H(I),$$

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

$$P = f_2(U) \oplus T \text{ and } Q = f_1(P) \oplus U$$

to the host.

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

3.3 Remark

Neither the amount of communication between host and card, nor the amount of work on the size of the card depend on the block size B of the full cipher. Thus, if B is not too small compared to the parameter b, which defines the size of the blocks sent to the card, the RaMaRK scheme is efficient. The card itself operates on $2b$ bit data blocks, and both $3b$ bit of information enter and leave the card. In practice, $b \geq 160$ gives a high level of security, while B can be huge.

4 Extended Security Requirements

Regarding the RaMaRK scheme, the authors of [3] pointed out that an adversary A who had access to card and lost the access again, can later chose special plaintexts where A can predict *a part of* the ciphertext. This makes it easy for A to distinguish between RaMaRK encryption and encrypting randomly.[2] Thus, according to the definition of [3], the RaMaRK scheme is not pseudorandom.

We believe that it is possible to extend the RaMaRK scheme to make it pseudorandom even in the sense of [3], i.e., with respect to insider adversaries. So far, this is an open problem. Note that all schemes in [3] are pseudorandom as defined there, but depend on *pseudorandom permutations* (i.e., block ciphers) – and thus are designed for smart cards with builtin encryption.

5 Implementation of Building Blocks on the Host

On the side of the host, we need standard cryptographic primitive operations, which can easily be implemented or found in a cryptographic function.

5.1 Hash Functions.

To combine the big block of data with the small blocks in the card we need a collision-free hash function. The calculation is performed on the host, so we can simply chose a well-tested hash fuction like SHA-1[5] or RIPE-MD160[4]. Both produce a 160-bit output, which seems to provide sufficient security.

[2] The intermediate value X only depends on the (P, Q)-part of the plaintext, and the encryption of the R-part only depends on X. If A chooses a plaintext (P, Q, R), having participated before in the encryption of (P, Q, R'), with $R \neq R'$, the adversary A can predict the C-part of the ciphertext corresponding to (P, Q, R) on her own.

5.2 Pseudo Random Bitgenerators.

In [8] the use of a stream cipher was suggested. We can also use a well-tested block cipher in the OFB or CFB mode (E.g. CAST-5 performs very fine even on small packets [10]).

6 Keydependent Pseudorandom Mappings on the Card

In this section we want to discuss how to realise Pseudo Random Mappings (PRM) with an Non-Encrypting smartcard. For the purposes of this paper, we suggest to use hash-based *Message Authentication Codes* (MACs) as tools. We specifically recommend the HMAC-construction from Bellare, Canetti, and Krawczyk [1], which is provably secure.

Note that a cryptographic hash function is defined to take a bit-sting of an arbitrary length as input, to produce a fixed-size bit-string as output. (In addition to this, it also has to satisfy some cryptographic security criteria.)

6.1 Using a Hash-Based MAC to realize PRMs

Trusting in a well-studied dedicated hash function, such as SHA-1 or RIPEMD-160, to realize a keydependent Message Authentication Code provides a couple of advantages for our scheme:

- Cryptographic hash functions have been well studied.
- Cryptographic hash functions are usualy faster than encryption algorithms.
- MACs based on SHA-1 or RIPE-MD160 mostly provide 160-bit output. So even birthday attacks which need 2^{80} operations are infeasible.
- Some hash-based MACs are provably secure if the underlying hash is secure.
- The proof of security for some MAC constructions can rely on quite weak assumptions on the hash function's security, compared to the standard assumptions for hash functions. Thus, even if the hash function we use is broken and insecure for signatures or other applications, it may still be infeasible to break the HMAC instantiated with this hash function.
- In many countries, it is more easy to export or import an authentication tool, such as a signature smart card, than to export or import an encrypting device, such as a smart card with a builtin encryption function.

6.2 HMAC: A Construction for Hash-Based MACs

HMAC [1] has the advantage that we can use any cryptographic hash function $\mathcal{H}$ as blackbox. The only restriction on $\mathcal{H}$ is the following: $\mathcal{H}$ is assumed to be an *iterative* hash function. That means, it internally uses a compression function, iteratively taking a fixed-size value as input (say, 512 bit), to produce a smaller-sized output (e.g., 160 bit). Most cryptographic hash functions known today are iterative. If needed, one can easily define a secure iterative hash functions based on a secure non-iterative one.
The HMAC function is defined like this:

$$\mathrm{HMAC}_K(x) := \mathcal{H}(\bar{K} \oplus \mathsf{opad} || \mathcal{H}(\bar{K} \oplus \mathsf{ipad} || x))$$

with $\mathsf{ipad} := \mathsf{0x36}$ repeated 64 times and $\mathsf{ipad} := \mathsf{0x5C}$ repeated 64 times[3], $\bar{K}$ is generated by appending zeros to the end of K to create a 64 byte string[4]. (Note that the specific values of ipad and opad are relevant for actually implementing HMACs without creating incompatible versions, but with respect to the security of HMACs, one mainly has to keep in mind $\mathsf{ipad} \neq \mathsf{opad}$.)

In [1] a proof is given, that the HMAC construction is secure against collision attacks and forgery attacks.

[3] The number of repetitions may actually change, depending on the input size of the underlying compression function. Most present-day hash functions, including the well-studied SHA-1 and RIPEMD-160, use a compression function with an input size of 512-bit (i.e., 64 byte).

[4] This size of 64 byte also changes with the input size of the compression function

As usual in present-day cryptography, the proof of security is based on some unproven but reasonable assumptions. The weaker such assumptions are, the stronger is the proof. It is thus remarkable, that the proof in [1] only makes very weak assumptions on the security of the underlying hash function (and no assumptions otherwise).

Consider selecting a hash function $\mathcal{H}$ for the HMAC construction, i.e., *instantiating* the HMAC construction with $\mathcal{H}$. Of course, this has to be done with great care. But it is the state-of-the-art in today's cryptography, that no one can rule out completely that this hash function $\mathcal{H}$ is later found to be insecure, e.g., collisions for $\mathcal{H}$ are found. A collision consists of two bit-strings $x \neq y$ with the same values, i.e., $\mathcal{H}(x) = \mathcal{H}(y)$. Such collisions do *exist* of course, but if it is feasible to actually *find* such collisions, this would be deathly for using $\mathcal{H}$ in the context of signatures. On the ohter hand, due to the weak assumptions on $\mathcal{H}$ the HMAC-construction requires, even a collision-prone hash function $\mathcal{H}$ could still satisfy the security requirements for HMACs, and HMACs instantiated with $\mathcal{H}$ could be secure, nevertheless.

References

[1] M. Bellare, R. Canetti and H. Krawczyk, "Keying hash functions for message authentication" (1996), in: Crypto 96, Springer LNCS.

[2] Blaze, M., "High-Bandwidth Encryption with Low-Bandwidth Smartcards", in: Fast Software Encryption (ed. D. Gollmann) (1996), Springer LNCS 1039, 33–40.

[3] Blaze, M., Feigenbaum, J., and Naor, M., "A Formal Treatment of Remotely Keyed Encryption", in: Eurocrypt '98, Springer LNCS 1403, 251-265.

[4] Dobbertin, H., Bosselaers, A., Preneel, B., "RIPEMD-160, a strengthened version of RIPEMD", Proc. of Fast Software Encryption (ed. D. Gollmann), LNCS 1039, Springer, 1996, pp. 71-82.

[5] NIST, "Secure Hash Standard", Washington D.C., April 1995.

[6] Luby, M., Rackoff, C., "How to construct pseudorandom permutations from pseudorandom functions", SIAM J. Comp., Vol 17, No. 2, 1988, pp. 239-255.

[7] Lucks, S., "Faster Luby-Rackoff ciphers", in: Fast Software Encryption (ed. D. Gollmann) (1996), Springer LNCS 1039.

[8] Lucks, S., "On the Security of Remotely Keyed Encryption", in: Fast Software Encryption (ed. E. Biham) (1997), Springer LNCS 1267.

[9] Lucks, S., "Accelerated Remotely Keyed Encryption", to appear in: Fast Software Encryption (1999), (ed. L. Knudsen) Springer LNCS, 1999.

[10] Weis, R., Lucks, S., "The Performance of Modern Block Ciphers in JAVA", to appear in: CARDIS'98, Springer LNCS.