Accelerated Remotely Keyed Encryption

Stefan Lucks*

Theoretische Informatik
University of Mannheim, 68131 Mannheim A5, Germany
lucks@th.informatik.uni-mannheim.de

Abstract. Remotely keyed encryption schemes (RKESs) support fast encryption and decryption using low-bandwidth devices, such as secure smartcards. The long-lived secret keys never leave the smartcard, but most of the encryption is done on a fast untrusted device, such as the smartcard's host.
This paper describes an new scheme, the length-preserving "accelerated remotely keyed" (ARK) encryption scheme and, in a formal model, provides a proof of security. For the sake of practical usability, our model avoids asymptotics.
Blaze, Feigenbaum, and Naor gave a general definition for secure RKESs [3]. Compared to their length-preserving scheme, the ARK scheme is more efficient but satisfies the same security requirements.

1 Introduction

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*. We think of the host being a computer under the risk of being taken over by an adversary, while the card can be a (hopefully tamper-resistant) smartcard, used to protect the secret key. We do not consider attacks to break the tamper-resistance of the smartcards itself. The host knows plaintext and ciphertext, but only the card is trusted with the key.

An RKES consists of two protocols, one for *encryption* and one for *decryption*. Given a β-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. This exchange of values can be iterated. During one run of a protocol, every challenge value may depend on the input and the previously given response values, and the response values may depend on the key and the previous challenge values. (In this paper, we disregard probabilistic RKESs, where challenge and/or response values also may depend on random coin flips.)

* Supported by German Science Foundation (DFG) grant KR 1521/3-1.

L. Knudsen (Ed.): FSE'99, LNCS 1636, pp. 112–123, 1999.

1.1 History

The notion of *remotely keyed encryption* is due to Blaze [2]. Lucks [5] 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 randomly, 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 smartcard, 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.

1.2 Pseudorandomness – Towards a Better Definition

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 the host and the 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 smartcard, 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.

Recently, Blaze, Feigenbaum, and Naor [3] found a better formalism to define the pseudorandomness of RKESs. Their idea is based on the adversary gaining direct access to the card for a certain amount of time, making q_h interactions with the card. For the adversary having lost direct access to the card, the encryption function should behave randomly. An attack is divided into two phases:

1. During the *host phase* (h-phase), the adversary is an insider, sends challenge values to the card and learns the card's response values. She may run through the en- and the decryption protocol and may also deviate from the protocol (note though, that the card always interprets the "next value" it reads as the next challenge value, until the current protocol is finished).
   At the end of the h-phase, the adversary loses direct access to the card, i.e., is no longer an insider.
2. In the *distinguishing phase* (d-phase), the adversary chooses texts and asks for their en- or decryptions. The answers to these queries are either chosen randomly, or by honestly en- or decrypting according to the RKES.
   The adversary's task is to distinguish between the random case and honest encryption.

Consider an adversary having encrypted the plaintext P^* and learned the corresponding ciphertext C^* during the h-phase. If she could ask for the encryption of P^* or the decryption of C^* during the d-phase, her task would be quite easy. In the d-phase, we thus need to "filter" texts that appeared in the h-phase before. But since the adversary may deviate from the protocol, it is not easy to formally define which texts are to be filtered out. The authors of [3] require an arbiter algorithm B to sort out up to q_h texts. This algorithm need not actually be implemented, it simply needs to exist. (The formal definition below looks quite complicated. Readers with few interest in formalisms should keep in mind that the arbiter B treats the special case that in the d-phase the adversary A asks for values already known from the h-phase. As will become clear below, the arbiter B for our scheme does exist and actually is quite simple.)

Throughout this paper, "random" always means "according to the uniform probability distribution". By $x \oplus y$ we denote the bit-wise XOR of x and y.

After that much discussion, we give the formal definitions (which are not much different from the ones in [3]).

1.3 Definitions

A (length-preserving) RKES is a pair of protocols, one for en- and one for decryption, to be executed by a host and a card. The length of a ciphertext is the same as that of the corresponding plaintext.

Let B be an algorithm, the "arbiter algorithm", which is initialized with a transcript of the communication between host and card during the h-phase.

During the *host phase* (h-phase), A may play the role of the host and execute both the card's protocols up to q_h times, together. A may send challenge values to the card not generated according to the protocol and does learn the corresponding response values.

During the *distinguishing phase* (d-phase), A chooses up to q_d texts T as queries and asks for the corresponding en- or decryptions.

W.l.o.g., we prohibit A to ask equivalent queries, i.e., to ask twice for the encryption of T, to ask twice for the decryption of T, or to ask once for the encryption of a T and some time before or after this for the decryption of the corresponding ciphertext. (Encrypting under a length-preserving RKES is a permutation, hence A doesn't learn anything new from asking equivalent queries.)

Before the d-phase starts, a switch S is randomly set either to 0 or to 1. If the arbiter B acts, A's query is answered according to the RKES; B can act on most q_h queries.

Consider the queries B does not act on. The answers are generated depending on S. Consider A asking for the en- or decryption of a text $T \in \{0,1\}^\beta$ with $\beta > a$. If $S = 0$, the response is evaluated according to the RKES. If $S = 1$, the response is a random value in $\{0,1\}^\beta$.

At the end of the d-phase, A's task is to guess S. A's advantage adv_A is

$$\mathrm{adv}_A = \Big|\ \mathrm{prob}[\text{“}A \text{ outputs } 1\text{”} \mid S = 1] - \mathrm{prob}[\text{“}A \text{ outputs } 1\text{”} \mid S = 0]\ \Big|$$

By q_h, we denote the number of interactions between the adversary A and the card during the h-phase, by q_d we denote the number of queries A asks during the d-phase; $q := q_h + q_d$ denotes the total query number.

A RKES is (t, q, e)-secure, if there exists an arbiter algorithm B such that any t-time adversary A with a total query number of at most q has an advantage of at most e.

1.4 Building Blocks and Security Assumptions

In this section, we describe the building blocks we use for our scheme. As will be proven below, our scheme is secure if its building blocks are secure. Note that definitions of standard cryptographic and complexity theoretic terms are left out here; they can be found e.g. in [4].

By a and b with $b \geq a$, we denote the blocksizes of our building blocks (while our scheme itself is able to encrypt blocks which may grow arbitrarily large). **Note that a and b are important security parameters!** We may use, say, a 64-bit block cipher such as triple DES as pseudorandom permutation, but this has significant consequences for the security of our scheme, even if the adversary cannot break triple DES.

Our **building blocks** are

- an a-bit blockcipher E (i.e., a family of pseudorandom permutations E_K over $\{0,1\}^a$),
- a family of pseudorandom functions $F_K\{0,1\}^b \longrightarrow \{0,1\}^a$ (F may be a b-bit blockcipher, if $a < b$ we ignore the last $b - a$ bits of the output),
- a hash function $H : \{0,1\}^* \longrightarrow \{0,1\}^b$, and
- a length-preserving stream cipher $S : \{0,1\}^* \longrightarrow \{0,1\}^*$, depending on an a-bit key. In practice, S may be an additive stream cipher, i.e., a pseudorandom bit generator where each bit of the output is XOR-ed with the plaintext (or ciphertext, if we think of S^{-1}). Just as well, S may be designed from the block cipher E, using a standard chaining mode such as CBC.

For the analysis, we assume our building blocks (such as block ciphers) to behave like their ideal counterparts (such as random permutations). This "ideal world" view allows us to define the resistance of our scheme against adversaries with unbound running time:

> A RKES is (q, e)-secure, if any adversary A with a query-complexity of at most q has an advantage of at most e.

Consider our RKES being (q, e)-secure in the ideal world, but not $(t, q, e+\epsilon)$-secure in the real world. If either t is large enough to be infeasible or $\epsilon \geq 0$ is small enough to be negligible, the notion "(q, e)-secure" can still approximatively describe the scheme's true security. Otherwise, we have found an attack on (at least) one of the underlying building blocks. Being (q, e)-secure for reasonable values of q and e implies that the construction itself is sound.

This is a standard argument for many cryptographic schemes, being composed from other cryptographic schemes and "provably secure".

Our **security assumptions** are

1. E_K is a random permutation over $\{0,1\}^a$, and for $K \neq K'$ the permutations E_K and $E_{K'}$ are independent.
2. $F_K\{0,1\}^b \longrightarrow \{0,1\}^a$, is a random function, i.e., a table of 2^b random values in $\{0,1\}^a$. Similarly to above, two random functions depending on independently chosen keys are assumed to be independent.
3. H is collision resistant, i.e., the adversary does not know and is unable to find a pair $(V, V') \in \{0,1\}^*$ with $V \neq V'$ and $H(V) \neq H(V')$ if $V \neq V'$.
4. S_K is a length-preserving stream cipher, depending on a key $K \in \{0,1\}^a$. I.e., for every number n, every plaintext $T \in \{0,1\}^n$, every set of keys $L = \{K_1, \ldots, K_r\} \subseteq \{0,1\}^a$ and every key $K \in \{0,1\}^a$, $K \notin L$, the value $S_K(T) \in \{0,1\}^n$ is a random value, independent of T, $S_{K_1}(T)$, ..., $S_{K_n}(T)$. Similarly, the value $S^{-1}(T)$ is a random value, independent of $S^{-1}_{K_1}(T)$, ..., $S^{-1}_{K_n}(T)$.

We do not specify the key sizes of E and F. We implicitly assume the security level of E and F (and thus their key size) to be long enough that breaking either of them is infeasible.

In the world of complexity theoretical cryptography, the usage of asymptotics is quite common. While this may simplify the analysis, it often makes the results less useful in practice. From a proof of security, the implementor of a cryptographic scheme may conclude the scheme to be secure if the security parameters are chosen large enough – but such a result provides little help to find out how large is "large enough". (Often, the implementor can find this out by very diligently reading and understanding the proof, though.)

This paper avoids asymptotics. If we call an amount of time to be "infeasible", we are talking about a *fixed amount of computational time*. What actually is considered infeasible depends on the implementors/users of the scheme and their threat model. Similarly, we use the word "negligible".

2 The ARK Encryption Scheme

Using the above building blocks, we describe the accelerated remotely keyed (ARK) encryption scheme. For the description, we use two random permutations E_1, E_2 over $\{0,1\}^a$ and two random functions $F_1, F_2 : \{0,1\}^b \longrightarrow \{0,1\}^a$. In practice, these components are realized pseudorandomly, depending on four different keys.

The encryption function takes any β-bit plaintexts, encrypts it, and outputs a β-bit ciphertext. The blocksize β can take any value $\beta \geq a$.

We represent the plaintext by (P, Q) with $P \in \{0,1\}^a$ and $Q \in \{0,1\}^{\beta-a}$; similarly we represent the ciphertext by (C, D) with $C \in \{0,1\}^a$ and $D \in \{0,1\}^{\beta-a}$. For the protocol description, we also consider intermediate values $X, Z \in \{0,1\}^b$ and $Y \in \{0,1\}^a$. The encryption protocol works as follows:

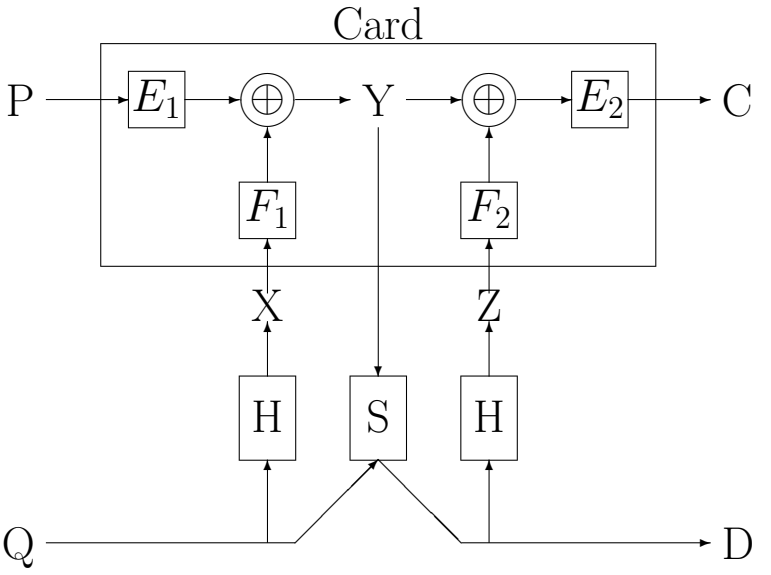

Fig. 1. The ARK encryption protocol.

1. Given the plaintext (P, Q), the host sends P and $X := H(Q)$ to the card.
2. The card responds with $Y := E_1(P) \oplus F_1(X)$.
3. The host computes $D := S_Y(Q)$.
4. The host sends $Z := H(D)$ to the card.
5. The card responds with $C := E_2(Y \oplus F_2(Z))$.

Decrypting (C, D) is done like this:

1. The host sends C and $Z = H(D)$ to the card.
2. The card responds with $Y = E_2^{-1}(C) \oplus F_2(Z)$.
3. The host computes $Q = S_Y(D)$.
4. The host sends $X = H(Q)$ to the card.
5. The card responds with $P = E_1^{-1}(Y \oplus F_1(X))$.

Note that by first encrypting any plaintext (P, Q) under any key and then decrypting the result the ciphertext under the same key, one gets (P, Q) again.

3 The Security of the ARK Scheme

By P_i, X_i, Y_i, Z_i, and C_i we denote the the challenge and response values of the i-th protocol execution, which may be either a en- or a decryption protocol. The protocol can either be executed in the h-phase, indicated by $i \in \{1, \ldots, q_h\}$, or in the d-phase, indicated by $i \in \{q_h + 1, \ldots, q\}$. A value Y_i is "unique", if $Y_i \notin \{Y_1, \ldots, Y_{i-1}, Y_{i+1}, \ldots, Y_q\}$. The ARK schemes security greatly depends on the values Y_k in the d-phase being unique, except when B acts ("in the d-phase"

indicates $k > q_h$). If the k-th en- or decryption query is answered according to the ARK scheme, and if Y_k is unique, then the answer is almost a random value.

Theorem 1. *For every number q and $e = 1.5 * q^2/2^a$, the ARK scheme is a (q, e)-secure length-preserving RKES.*

Proof. For the proof, we first define the arbiter algorithm, and then we bound the advantage of the adversary.

The arbiter algorithm B:

For $i \in \{1, \ldots, q_h\}$, the arbiter B compiles a list L_1 of all the pairs (P_i, X_i) and another list L_2 of all pairs (C_i, Y_i). These can be deduced from the transcript.

If, in the d-phase, A asks for the encryption of a plaintext (P_j, Q_j) with $j \in \{q_h + 1, \ldots, q\}$, the first challenge value for the card is the pair (P_j, X_j) with $X_j = H(Q_j)$. Only if the pair (P_j, X_j) is contained in the list L_1, B acts on that query, and the answer is generated according to the encryption protocol. Similarly, if A asks for the decryption of a ciphertext (C_j, D_j) in the d-phase, and if the corresponding challenge value (C_j, Z_j) is contained in L_2, B acts and the answer is generated according to the decryption protocol.

Now we argue, that B does not act on more than q_h queries. Due to Assumption 3, i.e., due to the collision resistance of H, the adversary A does not know more than one value Q with $H(Q) = X_i$. For the same reason, A does not know more than one value D with $H(D) = Z_i$. Hence for every $i \in \{1, \ldots, q_h\}$, A can ask no more than one plaintext (P_j, Q_j) to be encrypted during the d-phase, where the corresponding pair (P_j, X_j) would be found in the list L_1. Similarly, we argue for decryption queries. Finally, consider a plaintext $T = (P_j, Q_j)$ and the corresponding ciphertext $T' = (C_j, D_j)$. Asking for the encryption of T is equivalent to asking for the decryption of T', and we only regard non-equivalent queries. We observe: $((P_j, H(Q_j))$ is in the list $L_1) \Leftrightarrow ((C_j, H(D_j))$ is in $L_2)$.

The advantage of A:

In short, the remainder of the proof is as follows:

> Both for $S = 0$ and $S = 1$, we define what it means for the the k-th query ($k \in \{q_h + 1 \ldots, q\}$) to be "$\text{BAD}_k$". We define sets U_k and show that if query k is not BAD_k, the response is a uniformly distributed random value in U_k. Further, we evaluate the probability that the k-th query is BAD_k. We write BAD^* if any query k in the d-phase is BAD_k. If not BAD^*, then all the answers to A are randomly chosen according to a probability distribution induced by the sets U_k, and thus do not depend on S. This allows us to bound the advantage of A:
>
> $$\text{adv}_A \leq \text{prob}[\text{BAD}^* | S = 0] + \text{prob}[\text{BAD}^* | S = 1].$$

Let $k > q_h$ and A be asking for the encryption of a plaintext $(P_k, Q_k) \in \{0,1\}^a \times \{0,1\}^{\beta - a}$.

We assume $(P_k, X_k) \notin \{(P_1, X_1), \ldots, (P_{k-1}, X_{k-1})\}$. If $j \in \{1, \ldots, q_h\}$ and $(P_j, X_j) = (P_k, X_k)$, then B acts and the answer to this query does not depend

on S. If $j \in \{q_h+1 \ldots, k-1\}$, then $(P_j, Q_j) \neq (P_k, Q_k)$, because the j-th and the k-th query are not equivalent. If $(P_j, Q_j) \neq (P_k, Q_k)$, then $(P_j, X_j) = (P_k, X_k)$ would indicate a collision $Q_j \neq Q_k$ for H, something we assume A can't find.

Depending on previous protocol executions and responses, we define the set $U_k \subseteq \{0,1\}^a \times \{0,1\}^{\beta-a}$ of ciphertexts:

$$U_k := \big(\{0,1\}^a - \{C_1, \ldots, C_{k-1}\}\big) \times \big(\{0,1\}^{\beta-a}\big).$$

For $S = 1$, the ciphertext (C_k, D_k) is a uniformly distributed random value in $\{0,1\}^a \times \{0,1\}^{\beta-a}$. We define $\text{BAD}_k :\Leftrightarrow C_k \in \{C_1, \ldots, C_{k-1}\}$. Obviously, if not BAD_k, then (C_k, D_k) is a uniformly distributed random value in U_k. Further:

$$\text{prob}[\text{BAD}_k | S = 1] \leq \frac{k-1}{2^a}.$$

Now, we concentrate on $S = 0$. Here, we define

$$\text{BAD}_k :\Leftrightarrow \big(Y_k \in \{Y_1, \ldots, Y_{k-1}\} \text{ or } C_k \in \{C_1, \ldots, C_{k-1}\}\big)$$

Obviously, if not BAD_k, then $(C_k, D_k) \in U_k$. Also, if not BAD_k, then Y_k is not in $\{Y_1, \ldots, Y_{k-1}\}$, and then $S_{Y_k}(Q)$ is a uniformly distributed random value in $\{0,1\}^{\beta-a}$. Further, if $Z_j = Z_k$, then due to $Y_j \neq Y_k$, we have $C_j \neq C_k$. Apart from this restriction, C_k is a uniformly distributed random value in $\{0,1\}^a$, and especially: if not BAD_k, the ciphertext (C_k, D_k) is uniformly distributed in U_k.

If $X_j \neq X_k$, then $F_1(X_j)$ and $F_1(X_k)$ are two independent random values in $\{0,1\}^b$, and so are Y_j and Y_k. If $(P_k, X_k) \notin \{(P_1, X_1), \ldots, (P_{k-1}, X_{k-1})\}$ for every $j \in \{1, \ldots, k-1\}$, we have $P_j \neq P_k$ if $X_j = X_k$. In this case $Y_j \neq Y_k$. Hence $\text{prob}[Y_j = Y_k] \leq 2^{-a}$. Similarly, we get $\text{prob}[C_j = C_k | Y_j \neq Y_k] \leq 2^{-a}$. This gives

$$\text{prob}[\text{BAD}_k | S = 0] \leq 2\,\frac{k-1}{2^a}.$$

Thus:

$$\text{prob}[\text{BAD}^* | S = 1] \leq \sum_{k=q_h+1}^{q} \text{prob}[\text{BAD}_k | S = 1] \leq \frac{1}{2}\,\frac{q^2}{2^a},$$

and

$$\text{prob}[\text{BAD}^* | S = 0] \leq \sum_{k=q_h+1}^{q} \text{prob}[\text{BAD}_k | S = 0] \leq \frac{q^2}{2^a}.$$

Due to the symmetric construction of the ARK encryption protocol, the same argument applies if A, as the k-th query, asks for the decryption of the ciphertext (C_k, D_k) instead of asking for an encryption. Hence, the advantage of A is

$$\text{adv}_A \leq \frac{3}{2} * \frac{q^2}{2^a}.$$

□

4 The BFN Scheme [3]

In [3], the Blaze, Feigenbaum, and Naor describe a length-preserving RKES which we shortly refer to as the "BFN-scheme". As the ARK scheme is claimed to be *accelerated*, we need to compare the BFN scheme and the ARK scheme. Similarly to the ARK scheme, we represent the plaintext by (P, Q) with $P \in \{0,1\}^a$ and $Q \in \{0,1\}^{\beta-a}$, and the ciphertext by (C, D) with $C \in \{0,1\}^a$ and $D \in \{0,1\}^{\beta-a}$. Further, we consider $X, Z \in \{0,1\}^b$ and $I, J, Y \in \{0,1\}^a$. (Note that [3] only considers $b = a$.)

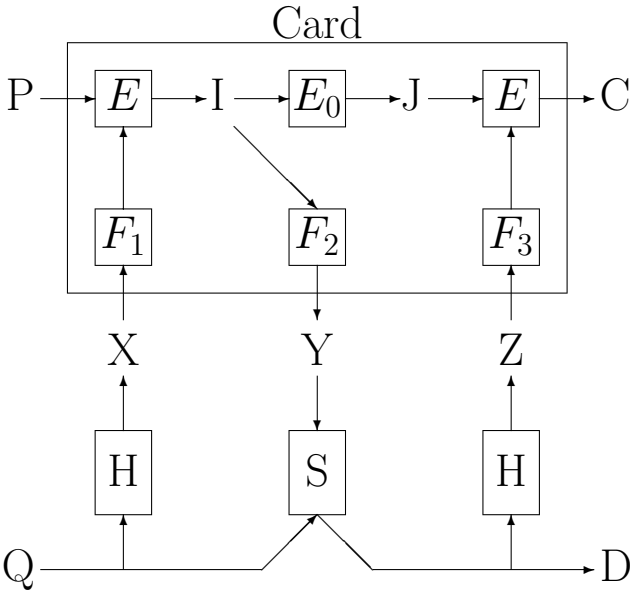

Fig. 2. The BFN encryption protocol [3].

As building blocks, we need one random permutation E_0 over $\{0,1\}^a$, three random functions $F_1, F_2 : \{0,1\}^b \to \{0,1\}^a$, $F_3 : \{0,1\}^a \to \{0,1\}^a$, and a block cipher E (i.e., a family of random permutations) E_K over $\{0,1\}^a$, depending on a key $K \in \{0,1\}^b$). The encryption protocol works as follows:

1. Given the plaintext (P, Q), the host sends P and $X := H(Q)$ to the card.
2. The card computes $X^* = F_1(X)$, and uses X^* as the key for E.
3. The card computes $I := E_{X^*}(P)$.
4. The card computes $J := E_0(I)$.
5. The card responds $Y := F_2(I)$ to the host.
6. The host computes $D := S_Y(C)$.
7. The host sends $Z := H(D)$ to the card.
8. The card computes $Z^* = F_3(Z)$, and uses Z^* as the key for E.

9. The card responds $C := E_{Z^*}(Y)$ to the host.

Decrypting (C, D) is done the obvious way.

Obviously, the ARK and the BFN scheme work quite similarly:

1. The cryptographic calculations on the side of the host are exactly the same for both protocols.
2. The communication between the host and the card is exactly the same for both protocols.
3. Inside the card, the ARK scheme needs four evaluations of cryptographic functions such as E_i and F_j. (Also, it needs two bit-wise XOR operations.) In contrast to this, the BFN scheme needs six evaluations of cryptographic functions.
4. Also inside the card, the ARK scheme allows the keys for the cryptographic functions to be chosen once and for all. On the other hand, the BFN scheme requires two evaluations of the block cipher E, where the keys are dynamically chosen, depending on some of the challenge values.

The third point indicates that, when implementing both schemes using the same building blocks, the BFN scheme's smartcard operations should take 50 % more time than the ARK scheme's smartcard operations. Due to the last point, things actually can be much worse for the BFN scheme. This greatly depends on the choice of the block cipher E and the key set-up time of E.

5 Final Remarks

5.1 The Importance of the Building Blocks

It should be stressed that our proof of security is with respect to any attack an adversary can come up with. Often, proofs of security in cryptography only deal with specific attacks, such as differential or linear cryptanalysis.

On the other hand, for practically using the ARK scheme, one has to instantiate the generic building blocks we are using. Our proof of security is only applicable, if all building blocks are secure, i.e., satisfy the security assumptions specified in this paper.

Hence, ARK scheme implementors have the freedom of choice to select their building blocks – and take the responsibility for the building blocks to be secure. If the building blocks are secure, the scheme is secure, too.

5.2 Inversion Security

This paper's reasoning is based on the security definition of Blaze, Feigenbaum and Naor [3], which seems to be more suitable than the one of Lucks [5]. It requires RKESs to be pseudorandom with respect to ex-insiders. Consider the RKES Σ being pseudorandom with respect to ex-insiders.

Clearly, Σ is pseudorandom with respect to outsiders, too. Also, Σ is forgery secure. Otherwise, the adversary could execute the protocol $q-1$ times in the h-phase to predict q plaintext/ciphertext pairs. In the d-phase, the adversary could ask for the encryptions of these q plaintexts, and compare the results with her own predictions. (Note that the arbiter B can act on at most $q-1$ plaintexts.)

But what about inversion security? Obviously, this property is quite desirable for some applications. Consider the RKES Σ^*, a simple modification of Σ. Both the en- and the decryption protocol of Σ^* start with an additional challenge value $\beta \in \{0,1\}$. The encryption protocol of Σ^* goes like this:

1. The host sends β to the card.
2. If $\beta = 0$, both parties follow the encryption protocol of Σ.
   Else, both parties follow the decryption protocol of Σ.

Similarly, we may define the decryption protocol of Σ^*. Note that we did change the protocol, but the en- and decryption functions remain the same. Since the additional challenge value just allows the adversary another way to execute the decryption protocol, the security of Σ^* and Σ is the same:

Theorem 2. *Σ is (q,e)-secure $\iff$ Σ^* is (q,e)-secure.*

On the other hand, Σ^* clearly is inversion insecure. If, e.g., we only allow the adversary to execute the encryption protocol of Σ^*, via $\beta = 1$ she can decrypt any ciphertext, because she can still run the decryption protocol of Σ.

Hence, inversion security is a property of its own right, not covered by the above notion of "(q,e)-security".

5.3 Implementing the ARK Scheme

For actually implementing the ARK scheme, one needs to instantiate the building blocks with cryptographic functions. In this context, the role of the blocksizes a and b are important. Note that the parameter b is simply ignored in the proof of security. But we require the hash function $H : \{0,1\}^* \to \{0,1\}^b$ to be collision resistant. Thus, it must be infeasible to do close to $\sqrt{2^b}$ *offline calculations*. On the other hand, due to Theorem 1 the ARK scheme's construction is sound if the total query number $q \ll \sqrt{2^a}$. This restricts the number of *online calculations* for the adversary.

The difference between offline calculations (using any hardware the adversary has money for) and online calculations on smartcards allows the implementor to chose $b > a$. The current author recommends $b \geq 160$ and $a \geq 128$.

The hash function H can be a dedicated hash function such as SHA-1 or RIPEMD-160. The block cipher E needs to be a 128-bit block cipher, such as the (soon to be chosen) DES-successor AES. In this case, the pseudorandom functions F_K must map a 160-bit input to a 128-bit output. One can realize these functions as message authentication codes (MACs), e.g. as CBC-MACs based on E. Such MACs are provably secure [1].

Finally, for the encryption function S we have a couple of choices. S may either be a dedicated stream cipher, such as RC4, or a block cipher in a standard

chaining mode, such as CBC or OFB. If one is using a block cipher, using the same one as inside the smartcard is reasonable.

Sometimes, a given application allows the security architect to drastically restrict the number of en- and decryptions during the lifetime of a key. If this number is well below 2^{32}, we may even use a 64-bit block cipher for E, e.g. triple DES. In this case, we need to observe two things:

1. The number of en- or decryptions should never exceed a previously defined bound $q^* \ll 2^{32}$, say $q^* = 2^{24}$. The card should be provided with a counter, to ensure the card to stop working after q^* protocol executions.
2. The encryption function S is defined to take an a-bit value as the key. Knowing one plaintext-part Q_j and the corresponding ciphertext part $C_j = S_{Y_j}(Q_j)$, the a-bit key Y_j can be found by brute force, on the average in 2^{a-1} steps. For $a = 64$, one should modify the ARK scheme and choose another key-dependent function to stretch the 64-bit value Y to get a larger key. E.g., think of using the block cipher E under a new key K_3, and send the 128-bit value $(Y_j, E_{K_3}(Y_j))$ to the card. This allows a key size for S of up to 128 bits. During the h-phase, the adversary learns q_h pairs of known plaintext Y_j and ciphertext $E_{K_3}(Y_j)$. This is no problem, since we anyway assume E to behave pseudorandomly.

References

1. M. Bellare, J. Kilian, P. Rogaway, "The Security of Cipher Block Chaining", in *Crypto 94*, Springer LNCS 839, 341–358.
2. M. Blaze, "High-Bandwidth Encryption with Low-Bandwidth Smartcards", in *Fast Software Encryption 1996* (ed. D. Gollmann), Springer LNCS 1039, 33–40.
3. M. Blaze, J. Feigenbaum, M. Naor, "A Formal Treatment of Remotely Keyed Encryption". in *Eurocrypt '98*, Springer LNCS.
4. M. Luby, *"Pseudorandomness and Cryptographic Applications"*, Princeton University Press, Princeton, 1996.
5. S. Lucks, "On the Security of Remotely Keyed Encryption", in *Fast Software Encryption 1997* (ed. E. Biham), Springer LNCS 1267, 219–229.