BEAST: A fast block cipher for arbitrary blocksizes

Stefan Lucks
Georg–August–Universität Göttingen
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 describes BEAST, a new blockcipher for arbitrary size blocks. It is a Luby-Rackoff cipher and fast when the blocks are large. BEAST is assembled from cryptographic hash functions and stream ciphers. It is provably secure if these building blocks are secure.

For smartcard applications, a variant BEAST-RK is proposed, where the bulk operations can be done by the smartcard's host without knowing the key. Only fast key-dependent operations remain to be done by the smartcard.

1 INTRODUCTION

Based on random functions, Luby and Rackoff (1988) described provably secure block ciphers. This theoretical break-through is of practical interest, since it enables us to assemble a secure cipher from secure components. Components are known, which we can reasonably expect to be secure. In this paper, the hash function SHA-1 (see Schneier, 1995) and the stream cipher SEAL (Rogaway and Coppersmith, 1993) are considered as components, though other choices would do, as well (Lucks, 1996). SHA-1 and SEAL have been suggested by Anderson and Biham (1996), but BEAST is faster than their ciphers.

BEAST, like Luby-Rackoff ciphers in general, is a Feistel cipher, similar to DES. While DES requires 16 rounds, BEAST only needs three. On the other hand, BEAST's round functions must be cryptographically stronger than the round functions of DES.

Due to its construction, BEAST performs best when operating on large blocks. This copes well with a possible use of BEAST in multimedia security applications, when a high throughput is required from the cipher, and the overall data volume is huge, too.

High-throughput encryption with smartcards is difficult, because of the smartcards' limited computational power and their slow communication links. Blaze (1996) suggested to share the encryption burden between the smartcard and its host. The smartcard's host does the bulk encryption work but is not trusted with the key, while the card only performs fast key-dependent operations. We exploit Blaze's idea for BEAST-RK, the remote-key variant of BEAST.

2 BACKGROUND

Encrypting with a block cipher means to apply a key-dependent permutation g to the plaintext, decrypting to apply the inverse g^{-1} to the ciphertext; g is computed by the 'encryption engine' and g^{-1} by the 'decryption engine'. A block cipher is secure if g appears like a randomly chosen permutation for anyone without knowledge of the key.

The type of attack we consider is a 'chosen plaintext attack', where the attacker chooses a plaintext x_1, injects x_1 into the encryption engine and gets the ciphertext $g(x_1)$. This is repeated with x_2, x_3, ... (see Figure 1). After a couple of plaintext injections the attacker has to decide whether g is a random permutation or not.

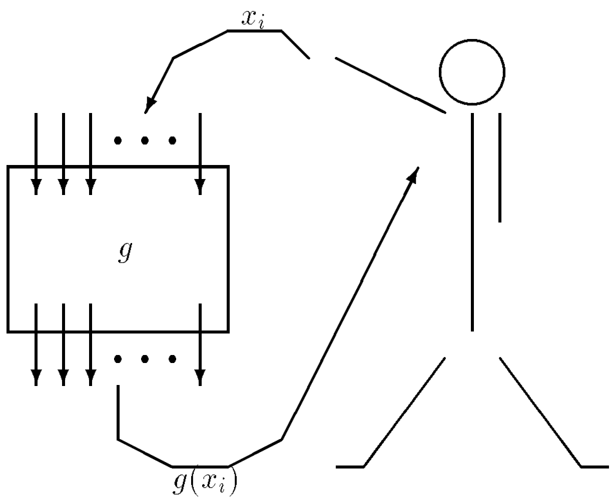

Figure 1 Chosen ciphertext attack.

Similarly one can consider chosen ciphertext attacks. We only consider block ciphers where the ciphertexts are as long as the plaintexts, so in this case, we can simply use the inverse permutation g^{-1} for encryption and the straight permutation g for decryption.

Resistance against such attacks is commonly accepted as a sufficient security criterion for block ciphers.

Every cipher is designed to be resistant against certain attacks, but fails to others. Among the advantages of using a cipher with a proof of security (under a reasonable assumption) is the simplicity of finding out which attacks the cipher is designed for – and which not. Overstretching the security of any cipher is like using an insecure one.

BEAST is not secure against 'combined chosen plaintext/chosen ciphertext attacks', where the attacker accesses both the encryption and the decryption engine. It is very dangerous, anyway, to allow the enemy to decrypt and encrypt with the secret key. **Look out for this type of attack and try to rule it out for your application – but if you can't, use another cipher!**

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

$$\begin{aligned} S &= L \oplus f_1(R), \\ T &= R \oplus f_2(S), \quad \text{and} \\ U &= S \oplus f_3(T). \end{aligned}$$

This way we have defined a permutation $\psi(f_1, f_2, f_3)(L, R) = (U, T)$ over $\{0,1\}^{l+r}$. This is represented by Figure 2 – just start with L and R and follow the arrows. Similarly, $\psi(f_3, f_2, f_1) = \psi^{-1}(f_1, f_2, f_3)$ is the inverse.

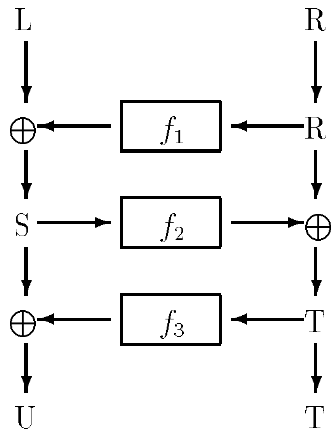

Figure 2 The permutation $\psi(f_1, f_2, f_3)(L, R) = (U, T)$.

Luby and Rackoff (1988) showed in their famous paper that $\psi(f_1, f_2, f_3)$ 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 is undistinguishable from a random permutation. Maurer (1992) gave an amazingly simple proof for Luby's and Rackoff's theorem. Neither Luby and Rackoff nor Maurer regarded $l \neq r$.

Theorem 1 *Let $g : \{0,1\}^{r+l} \longrightarrow \{0,1\}^{r+l}$ be either a random function or $g = \psi(f_1, f_2, f_3)$, 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), ..., (L_k, R_k) and receives the corresponding (U_1, T_1), ..., (U_k, T_k) with $(U_i, T_i) = g(L_i, R_i)$. Then

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

Informally spoken, there is no reasonable chance for an attacker to distinguish between $\psi(f_1, f_2, f_3)$ and a random function, except when the attacker has chosen close to $\sqrt{2^n}$ plaintexts and has got the corresponding ciphertexts. The proof is based on Maurer's proof for $l = r = n$ and is given by Lucks (1996).

Lucks also found a 'shortcut' for the third round: If $r > l$, we may replace the function f_3 by a function $f_* : \{0,1\}^l \longrightarrow \{0,1\}^l$ which only uses any l of the r input bits to f_3 and ignores the remaining $r - l$ bits. If $r \gg l$, one can expect to evaluate f_* much faster than f_3.

Theorem 2 *Let the function $f_* : \{0,1\}^l \longrightarrow \{0,1\}^l$ be a random function. If – except for $f_3(T) = f_*(T \bmod 2^l)$ – the conditions of theorems 1 are satisfied, then $|P_{\text{RAND}} - P_{\text{PERM}}| < \frac{k^2}{2^n}$.*

3 PSEUDORANDOMNESS AND 'SECURITY'

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. We know that if the pseudorandom functions are secure, the block cipher 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-Rackoff ciphers are 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, ..., x_k and gets the corresponding outputs $g(x_1)$, $g(x_2)$, ..., $g(x_k)$.'

Note that the computation of the k ciphertexts takes time k and is a part of A's overall run time.

4 THE BLOCK CIPHERS BEAR AND LION – AND BEAST

If $l \neq r$, 'compressing' (pseudo-)random functions f_i (with more input bits than output bits) and 'expanding' $f_{i\pm1}$ (less input bits than output bits) alternate in Figure 2. For compressing, cryptographic hash functions are well suited – Anderson and Biham (1996) suggested to use SHA-1. For expanding, they considered the stream cipher SEAL.

Cryptographic hash functions such as SHA-1 are authentication tools. We may use them as building blocks for our ciphers, but then we have to demand the hash functions to be pseudorandom. Being pseudorandom is a widely accepted security-related design goal for cryptographic hash functions, anyway. Anderson (1993) describes some risks of using non-pseudorandom hash functions for authentication.

SEAL is a stream cipher explicitly designed by its authors to be a pseudorandom function, too. (A stream cipher is 'secure', if given random inputs the outputs are undistinguishable from random outputs – where the output-size exceeds the input-size. Being a pseudorandom function requires more, since here the outputs have to be undistinguishable from random functions even if different inputs are deliberately chosen by the attacker.)

On a 133 MHz DEC Alpha machine (a 'sandpiper'), Roe (1994) measured a compression speed of about 40 Mbit/sec for SHA-1 and a expansion speed of more than 100 Mbit/sec for SEAL. The second result, though, is a more asymptotical one since SEAL runs through a very slow key-setup before it actually starts encryption. The key-setup takes about as much time as hashing a 32000 bit input with SHA-1 (i.e. as evaluating the internal 512 bit to 160 bit compression function of SHA-1 about 200 times – see Schneier (1995)).

Anderson and Biham proposed two block ciphers for flexible but large blocks: BEAR and LION, both similar to Figure 2 with $(l + r)$-bit Blocks. BEAR was based on the choice $l = 160 \ll r$, with two SHA-1 r-bit to l-bit compression steps and one SEAL l-bit to r-bit expansion step, similarly LION on $l \gg r = 160$ with two expansion step and one compression step. For large blocks (i.e. blocks greater than about 6 Kbyte), LION is faster than BEAR.

Anderson and Biham only considered a very weak type of attack, and their security proof for BEAR and LION is not valid for chosen plaintext attacks. But thanks to theorem 1 both ciphers are as secure as any block cipher based on Figure 2 – if the underlying pseudorandom functions are secure.

Theorem 2 enables us to define the BEAST ('Block Encryption Algorithm with Shortcut in the Third round', see Figure 3), a variant of BEAR, but faster than both BEAR and LION.

Since the round functions of Luby-Rackoff ciphers have to depend on a key*, we need keyed variants SHA_K and SEAL_K of SHA-1 and SEAL:

$$\mathrm{SHA}_K(x) = \mathrm{SHA}(K \oplus x) \quad \text{and} \quad \mathrm{SEAL}_K(x) = \mathrm{SEAL}(K \oplus x).$$

*Actually, the second round of BEAR and LION does not depend on a key. This is theoretically sound, but does not simplify or speed-up the ciphers significantly.

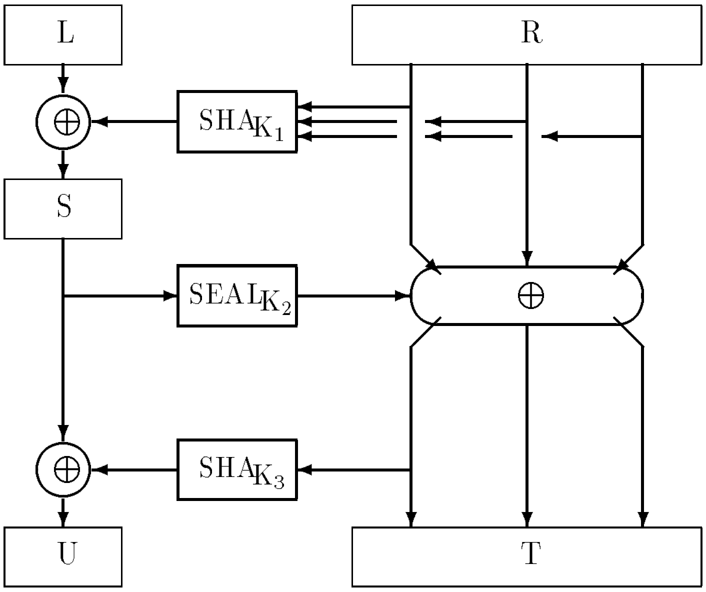

Figure 3 The block cipher BEAST.

Note that the input size to the keyed hash function is known in advance, otherwise we could use e.g. $\mathrm{SHA}_K(x) = \mathrm{SHA}(K||x||K)$, where '$||$' stands for the concatenation of bit strings. BEAST can be described by the following equations:

$$\begin{aligned} S &= L \oplus SHA_{K_1}(R), \\ T &= R \oplus SEAL_{K_2}(S), \quad \text{and} \\ U &= S \oplus f_3(T^*) \quad \text{with } T^* = \text{`first } l \text{ bits of } T\text{'}. \end{aligned}$$

Here, K_1, K_2, and K_3 represent the BEAST's round keys. It is straightforward to generate them from a smaller master key K_M using SEAL_{K_M}.

How fast is BEAST? On the same 'sandpiper' Roe used, Anderson and Biham measured the speed of BEAR and LION. With the blocksize 1 Mbit, they measured an encryption speed of 13.62 blocks/sec for BEAR and 18.68 blocks/sec for LION. BEAR consist of two rounds of hashing with SHA-1 and one round of SEAL, while in LION there are one SHA-1- and two SEAL-rounds. For BEAST, the encryption speed is dominated by one SHA-1- and one SEAL-round – the time required for the third round is negligible for large blocks. Based on these facts, we expect BEAST to encrypt and decrypt 1 Mbit blocks at about 23.6 blocks/sec on on the same computer. BEAST's expected 23.6 Mbit/sec can be compared with about 1.86 Mbit/sec of the classical 'DEA' (Roe, 1994).

5 REMOTE-KEY ENCRYPTION WITH BEAST-RK

If SHA-1 is collision-resistant and SHA_K is a secure pseudorandom function, then the function $F_K(x) = \mathrm{SHA}_K(\text{SHA-1(x)})$ is a secure pseudorandom function, too. Similarly, if SEAL_K also is a secure pseudorandom function, $G_K = \mathrm{SEAL}(\mathrm{SHA}_K(x))$ is so, as well. Is there a point in replacing a secure pseudorandom function by another secure but slower one?

There is – sometimes. Obviously, the operation which does the bulk work (if the blocks are large), i.e. SHA-1 in F_K and SEAL in G_K, does not depend on a key. The key-dependent operation is always SHA_K, operating on fixed input and output blocks of 160 bits – whatever the actual blocksize may be. This leads us to the idea of doing the bulk operation on a fast but untrusted host and doing the key-dependent operation on a slow tamperproof device like a smartcard.

This observation leads to BEAST-RK, see Figure 4. The r-bit right side R of the input is split into a 160-bit subblock R^* and a $(r-160)$-bit subblock R^{**}. Similarly, the second round key K_2 is split up into K_2^* and K_2^{**}, and the right side T of the output is the concatenation of a 160-bit subblock T^* and a $(r-160)$-bit subblock T^{**}. BEAST-RK operates as follows:

$$
\begin{array}{lcll}
S & = & L & \oplus \ \mathrm{SHA}_{K_1}(R^*||SHA(R^{**})) \\
T^* & = & R^* & \oplus \ \mathrm{SHA}_{K_2^*}(S) \\
T^{**} & = & R^{**} & \oplus \ \mathrm{SEAL}(\mathrm{SHA}_{K_2^{**}}(S)) \\
U & = & S & \oplus \ \mathrm{SHA}_{K_3}(T^*)
\end{array}
$$

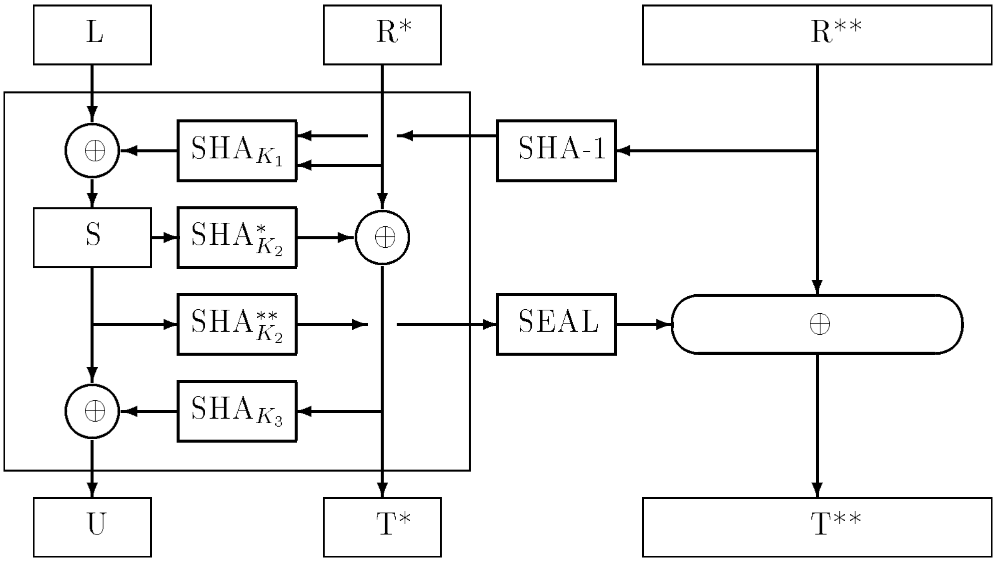

Figure 4 The remote-key variant BEAST-RK of BEAST.

In order to share the encryption between the host and the (hopefully tamperproof) card, the following protocol is used:

1. Given L, R^*, and R^{**}, the host sends L, R^* and SHA(R^{**}) to the card.
2. The card computes S, T^*, $\mathrm{SHA}_{K_2^{**}}(S)$, U and sends U, T^* and $\mathrm{SHA}_{K_2^{**}}(S)$ to the host. S remains secret for the host.
3. The host uses $\mathrm{SHA}_{K_2^{**}}(S)$ to compute T^{**}.

The large rectangle below the L- and R^*-box represents the smartcard. When encrypting one block, three 160-bit inputs L, R^*, and SHA-1(R^{**}) enter the smartcard, and three 160-bit outputs U, T^*, and $\mathrm{SHA}^{**}_{K_2}(S)$ leave it – independently of the blocksize.

BEAST-RK looks like is a four round cipher, but it is a three round one, just like BEAST. The output of the second round function is $(R^* \oplus T^*)||(R^{**} \oplus T^{**})$, the round key is $K_2 = K_2^*||K_2^{**}$.

During a chosen plaintext attack, the attacker not only gets ciphertexts, but also intermediate values of the form $\mathrm{SHA}^{**}_{K_2}(S)$. Does giving away such intermediate information weaken BEAST-RK, compared to BEAST?

The answer is 'no'. Let an attacker be given a plaintext block (L_0, R_0^*, R_0^{**}) and a bit-string X of block-length. For $i \in \{1, \ldots, k\}$, the attacker chooses plaintext blocks (L_i, R_i^*, R_i^{**}) different from (L_0, R_0^*, R_0^{**}) and gets the corresponding intermediate values $\mathrm{SHA}^{**}_{K_2}(S_i)$ and ciphertext blocks (U_i, T_i^*, T_i^{**}). If BEAST-RK's components SHA-1 and SEAL are secure, it is infeasible for the attacker to decide whether X is randomly generated X or $X = (U_0, T_0^*, T_0^{**})$. Here, we omit a formal proof but note that this can be proved based on the following three observations:

1. If the hash function SHA-1 is collision-resistant, it is infeasible find different plaintext blocks $(L_1, R_1^*, R_1^{**}) \neq (L_2, R_2^*, R_2^{**})$ with $L_1 = L_2$, $R_1^* = R_2^*$, and $\mathrm{SHA}(R_1^{**}) = \mathrm{SHA}(R_2^{**})$.
2. Consider a variant of BEAST-RK with 480-bit blocks, where SEAL is replaced by the identical function, i.e. $T^{**} = R^{**} \oplus \mathrm{SHA}^{**}_{K_2}(S)$. If BEAST is secure, then so is this variant. Hence, given any 'new' plaintext block (L_i, R_i^*, R_i^{**}), it is infeasible for attackers to distinguish between the corresponding $(U_i, T_i^*, \mathrm{SHA}^{**}_{K_2}(S_i))$ and a random triple $(X, Y, Z) \in (\{0,1\}^{160})^3$.
3. If SEAL is a secure stream cipher, attackers who don't know $\mathrm{SHA}^{**}_{K_2}(S)$ can't distinguish between $\mathrm{SEAL}(\mathrm{SHA}^{**}_{K_2}(S))$ and random bit strings.

IF BEAST is secure, then BEAST-RK is so, too. In some sense, BEAST-RK is more secure than BEAST, since BEAST-RK does not require SEAL do be a pseudo-random function. Any secure stream cipher could replace SEAL – possibly a stream cipher with a faster set-up.

BEAST-RK is somewhat similar to Blaze's RKEP, the 'remotely keyed encryption protocol' (Blaze, 1996). But, as outlined above, BEAST-RK is provably secure if its components are secure. No such result is known for the RKEP.

BEAST-RK can be used e.g. for pay-TV and similar applications, where encrypted data are transmitted via a broadcast-channel. Decryption is done on the customers' decoder boxes (these can either be hardware, or software running on PCs), connected to tamperproof tokens. The sender A lives from selling these tokens. Since one can not expect all of the customers to be honest, A is literally giving away decryption engines to potential attackers. In other words, A has to worry about chosen ciphertext attacks and should use inverse BEAST-RK for encryption and straight BEAST-RK for decryption.

The boxes and the tokens can also be used for public-key like encryption: The customers encrypt their messages using straight BEAST-RK, and only A can read the messages. Even token-owners who listened to an encrypted message can do no better than to guess the message and to verify their guess by encrypting it on their own. This feature is due to the smartcard and its one-way use – by no means can BEAST or BEAST-RK be regarded as public-key ciphers.

6 CONCLUDING REMARKS

As mentioned in section 2, one should look out for possible combined chosen plaintext/chosen ciphertext attacks. In the case of the above application example, there are two potential risks:

1. A encrypts and broadcasts data provided by the third party B.
2. A decrypts messages sent to her and publishes these.

If B is not considered trustworthy by A, there is but one cure for the first risk: A has to use another cipher.

If one does not need remote-key encryption, Anderson's and Biham's LIONESS, the four-round variant of LION, should be considered. LIONESS is slower than BEAR and LION, but secure against combined chosen plaintext/chosen ciphertext attacks. It is unclear whether BEAR, LION, or LONESS can be adapted for remote key encryption – as we have done with BEAST.

The second risk can be ruled out by a careful application design, so BEAST-RK can be used. Troublesome are those customers who – instead of choosing a message, encrypting and sending it – directly choose a ciphertext x and send it to A. If A gives away the decryption of such an x, she allows access to her decryption engine. Hence A must not reply something like 'your message has been scrambled, I only read ...'

The cure is to demand the messages to be redundant in a certain way and to ignore all other messages. If e.g. the last n bits of every message block have to be zeroes, a plaintext corresponding to a chosen ciphertext will be ignored with the probability $1 - 2^{-n}$.

The designers of new cryptosystems frequently prove security against certain types of cryptanalysis, e.g. differential or linear cryptanalysis. Here, we need no such proof. We know that our ciphers are secure against **all** types of cryptanalysis, even the ones not yet discovered, if the underlying building blocks are secure.

Note that we can use our cipher like a 'box of bricks', i.e. define variants of BEAST and BEAST-RK not based on SHA-1 and SEAL, but on other hash functions or stream ciphers. This may be done because of lack of trust in the security of SHA-1 and SEAL, or in order to to speed-up the block cipher. E.g. one can replace SEAL by another component with a faster set-up. Lucks (1996) considers a hash function based replacement for SEAL. There are two advantages of that proposal: It is faster if the blocksize is moderately large, and the security of the hash function is a sufficient security criterion for the security of the block cipher.

REFERENCES

R. Anderson (1993) The Classification of Hash Functions, in *Fourth IMA conference on cryptography and coding*, 83–93.

R. Anderson, E. Biham (1996) Two Practical and Provably Secure Block Ciphers: BEAR and LION, in *Fast Software Encryption* (ed. D. Gollmann), Springer LNCS 1039, 113–120.

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

M. Luby, C. Rackoff (1988) How to construct pseudorandom permutations from pseudorandom functions, *SIAM J. Computing*, Vol. 17, No. 2, 373–386.

S. Lucks (1996) Faster Luby-Rackoff Ciphers, in *Fast Software Encryption* (ed. D. Gollmann), Springer LNCS 1039, 189–203.

U. Maurer (1992) A Simplified and Generalized Treatment of Luby-Rackoff Pseudorandom Permutation Generators, in *EuroCrypt '92* (ed. R. Rueppel), Springer LNCS 658, 239–255.

P. Rogaway, D. Coppersmith (1993) A Software-Optimized Encryption Algorithm, in *Fast Software Encryption* (ed. R. Anderson), Springer LNCS 809, 56–63.

M. Roe (1994) Performance of Block Ciphers and Hash Functions – One Year Later, in *Fast Software Encryption* (ed. B. Preneel), Springer LNCS 1008, 359–362.

B. Schneier (1995) *Applied Cryptography*, Wiley.

BIOGRAPHY

Stefan Lucks received his Diplom (diploma) in computer science from the University of Dortmund. He expects to receive his Ph. D. soon. Currently, he is employed as a scientific assistant at the University of Göttingen, where he works on the design and analysis of efficient cryptographic schemes. Apart from cryptography, his research interests include computer security, communications security, and complexity theory.

This paper was published at:

P. Horster (ed.), IFIP Conference on Communications and Multimedia Security (1996), Chapman & Hall, 144–153.