Fast Encryption for Set–Top Technologies

Stefan Lucks[a], Rüdiger Weis[b], and Volker Hilt[b]

[a] Theoretische Informatik,
University of Mannheim,
68131 Mannheim, Germany
`lucks@th.informatik.uni-mannheim.de`

[b] Praktische Informatik IV,
University of Mannheim,
68131 Mannheim, Germany
`{rweis,hilt}@pi4.informatik.uni-mannheim.de`

ABSTRACT

In this paper we present two approaches to combine recent results of cryptographic research with the requirements of modern multimedia systems. The first is to evaluate modern block ciphers in a JAVA–environment. The second approach is based on recent developments regarding fast Luby–Rackoff ciphers. Paradoxically, it deals with doing "high-bandwidth encryption with low-bandwidth smartcards".[6] Also, we discuss implementation considerations for a specific multimedia project, the multimedia database for teleteaching at the University of Mannheim.

Keywords: Set–Top technology, modern block ciphers, performance, JAVA, Luby–Rackoff cipher, Remotely Keyed Encryption, GRIFFIN digital libraries, TeleTeaching.

1. INTRODUCTION

In many multimedia applications, such as pay-TV, between cryptographic security conflicts with efficiency. Application developers need secure encryption (or other cryptographic tasks such as secure authentication, identification, . . .). They also need fast encryption (authentication, identification, . . .) because multimedia applications often serve high-bandwidth channels, and because the computing device is needed to do other useful tasks apart from encrypting, e.g. mpeg en- or decoding.

Commercial demands and the technological requirements of set-top boxes make this conflict between security and efficiency even more difficult to resolve:

- The software should be portable to different hardware environments without much work. Thus a machine-independent high-level application of the software is desirable, which prohibits machine-specific optimizations.
- The customer, e.g. in pay-TV applications, is handed out a set-top box for their legitimate purposes, while this is a piece of hardware today, in the future it may be some software running on a sufficiently powerful PC. Implementing a given set-top box for different hardware must be keept simple and cost–efficient for the provider.
- The connection between the provider and the customer's set-top box needs cryptographic protection.

Frequently, cryptographic keys must be used by the legitimate customer as well as protected from cheating customers. Often, the only way to deal with this is to store the keys in a small device, connected to the set-top box and trusted by the provider, a smartcard. In this case, things become even more difficult:

- The smartcard is under the physical control of the customers, including potential cheaters. Hence it must be physically tamper resistant, and the cryptographic keys must never leave the smartcard.
- Due to the tamper resistance, a smartcard is always a slow device, compared to a set-top box.

Neverless the computational overhead for the cryptographic protection should not be too high.

Victimizing security to save on efficiency is always dangerous. The easy of availability of pirate decoders for many pay-TV standards and the resultant commercial damage should be a lesson to everyone who considers this. In this paper, we evaluate techniques for having both: efficiency and high cryptographic security.

2. MODERN BLOCK CIPHERS IN JAVA

The design of block ciphers may involve certain bit–fiddling operations best done in hardware or in assembler. If one restricts oneself to a certain high-level language, as we do, this has a great impact on the cipher's performance. On the other hand, a cipher needs only to be *fast enough* for a given application. If this can be done without low-level optimizations, then the application (or at least the cryptographic part of it) can be realized as a portable high-level implementation. Hence, we measure the efficiency of block ciphers in JAVA.

JAVA is a portable, object oriented programming language with many interesting security features (e.g. sandbox paradigm, bytecode verification). Initially, JAVA was designed for *set-top boxes*. Today, JAVA is highly portable and can be used in different environments like phones, computer networks, and, of course, in smartcards.

The JAVA Security API[45] from SUN is a widely known cryptographic package in JAVA. It is still under construction and provides standard interfaces for different cryptographic protocols. Other packages are from RSA[40] and Microsoft.[47] A freeware alternative to these packages without export restrictions is the Cryptix library,[9] which we use in our tests.

2.1. Ciphers

We concentrate on established ciphers, published at least a couple of years ago, carefully examined by the cryptographic community and widely used today. All ciphers we evaluate are block ciphers with 64-bit blocks.

2.2. DES Family

The famous DES[14] encryption algorithm, originally designed for confidential but non–classified data, is used in many applications today (e.g. electronic banking). Internally, the DES is based on the so–called Feistel structure, in which a simple round function is repeated several times (in the case of DES, exactly 16 times). This allows inexpensive hardware implementations of DES.

Since DES has been cracked by a brute force attack by the nonprofit organisation "EFF" in only 56 hours ([10] – see also[12]) we suggest to use Triple-DES[44] or DESX.[37] (The key length today should be at least 75–90 bits.[7])

Note that Triple DES is quite inefficient and makes suboptimal use of its 168 key bits. Recently, the effective key length of Triple DES was found to be no more than 108 bits.[30] This, of course, is still secure against brute force attacks.

2.3. IDEA

The IDEA block cipher[25] has a strong mathematical foundation and possesses good resistance against differential cryptanalysis.[24] The key length of 128 bit protects well against brute–force attacks. Many cryptographers considers IDEA to be the strongest public algorithm.[42] IDEA uses three algebraic operations: bit-wise XOR, addition modulo 2^{16} and multiplication modulo $2^{16}+1$. On small machines such as smartcards, the efficiency of IDEA greatly depends on the time required by such a multiplication. IDEA was the preferred cipher in the PGP versions (Pretty Good Privacy) until version 2.63.

2.4. Blowfish and CAST

In spite of having been developed independently, the basic structures of the block ciphers Blowfish[41] and CAST are very similar.[1] In the case of CAST we analyze CAST-128 which is proposed for standardisation.[2] While the size of a Blowfish key is variable (limited to no more than 448 bit in the 16–round version), a CAST-128 key has a fixed size of 128 bit.

CAST was designed with resistance to differential cryptanalysis, linear cryptanalysis and related–key cryptanalysis in mind. CAST possesses a number of other cryptographic advantages compared to DES, e.g. no complementation property and the absence of weak and semi-weak keys. No such strong results regarding known cryptanalytic techniques have been published for Blowfish. On the other hand key–dependent S-boxes seem to have fine resistance against unknown cryptanalytic attacks. To conclude no attacks of any practical relevance are known on either CAST or Blowfish.

Note, though, that an implementation of either cipher needs 4 kilobytes of memory for the S-boxes. The CAST S-boxes are fixed and can be stored in ROM, while the Blowfish S-boxes are key-dependent and hence need to be

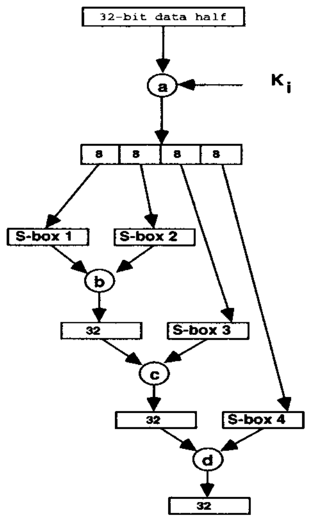


Figure 1. Generalized CAST–round, using operations a, b, c and d.[1]

stored in RAM. For smartcards, this is a significant disadvantage – memory, and especially RAM, is expensive on a smartcard.

The Blowfish key schedule is more involved than the CAST key schedule, hence for small blocks, CAST is much faster than Blowfish. For large blocks, our measurements indicate that Blowfish is faster than CAST. This has something to do with the CAST round function using key-dependent rotations, XOR-operations and additions/subtractions, while Blowfish only uses XOR and addition.

2.5. SAFER

SAFER[32] is a freely available block cipher which does *not* depend on the Feistel structure. It was designed with small processors (i.e., eight–bit microprocessors) in mind and thus is of special interest with respect to smartcards. We concentrate on the SAFER variant SAFER-SK 128/R 13 with an improved key schedule, 128 key bits, and 13 rounds. This variant is proposed for OpenPGP[33] standardization (e.g. dlb[16]).

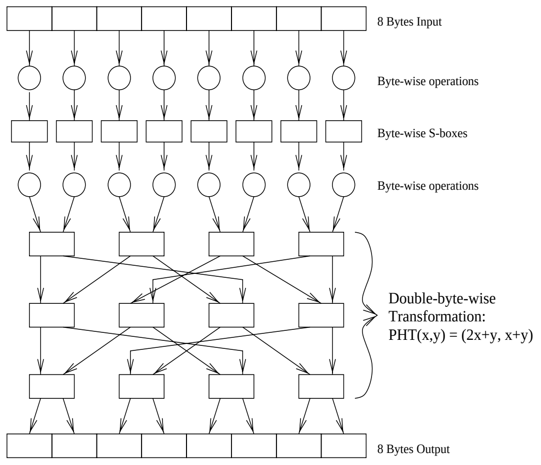


Figure 2. SAFER uses byte-oriented operations.

2.6. Performance

Publications exploring the performance of ciphers are surprisingly rare in the literature. Some years ago, Roe published benchmark results[35] and one year later, he published an update.[36] More recently, Schneier and Whiting counted the number of machine cycles some cryptographic operations needed on a Pentium.[43] They concentrated on low-level optimizations for a specific machine. This certainly is of great interest for application developers who need to write an application as fast as possible on that specific machine (and possibly on similar machines, too). But it does not help much to answer the question we consider in this paper.

Apart from its security, the efficiency of a block cipher is of main interest for application developers. Typically, block ciphers first run the *key schedule*, and then start encrypting. For the performance, two numbers are of importance: the *latency* and the *throughput.* Given key and plaintext, the latency is the time to wait until the first ciphertext block is known. The throughput is the speed at which huge plaintexts can be encrypted, ignoring the key schedule. (Depending on the actual application, either the latency or the throughput may be of greater importance.)

Essentially, the latency allows to estimate the speed at encrypting small plaintexts, and the throughput stands for the speed at encrypting large plaintext. In our experiments, we actually measure the speed (in kilobytes per second) at encrypting small plaintexts (of 1 kilobyte) and the speed at encrypting large plaintexts (of 100 kilobytes).

We concentrate on the *speed of encryption.* We did not find great differences to the speed of decryption, though the key schedule time for decryption could be somewhat different from its encryption counterpart.

Our machine-independent approach is mainly of interest to application developers who need efficient encryption, but don't need to push a given machine to its performance limits.

Our measurements are done on a conventional PC, not on a smartcard. Nevertheless, we believe the measurements to be of interest to smartcard developers, too. JAVA Smartcards are on their way, and the inherent advantages and limitations of the JAVA programming language are of relevance for these.

The test was performed on a PC with an Intel Pentium 200 MMX CPU under Linux 2.0.33. The main memory size was 64 megabytes, and we uses the SUN JAVA implementation JDK 1.1.5. All JAVA classes were compiled with the compiler-optimization *disabled* because otherwise some of the self-tests failed. For the same reason, no just-in-time compiler was used. No native routines were used although some are shipped with the current version of Cryptix and would give an important speedup. The test was performed on a PC with an Intel Pentium 200 MMX CPU under Linux 2.0.33. The main memory size was 64 megabytes, and we the SUN JAVA implementation JDK 1.1.5. All JAVA classes were compiled with compiler-optimization *disabled* because otherwise some of the self-tests failed. For the same reason, no just-in-time compiler was used. No native routines were used although some are shipped with the current version of Cryptix and would give an important speedup.

The test measured the encryption speeds of various algorithms depending on different plaintext sizes in ECB mode. Particularly, 1024 and 102400 byte plaintexts were used.

Table 1. Encryption speed in kByte/s

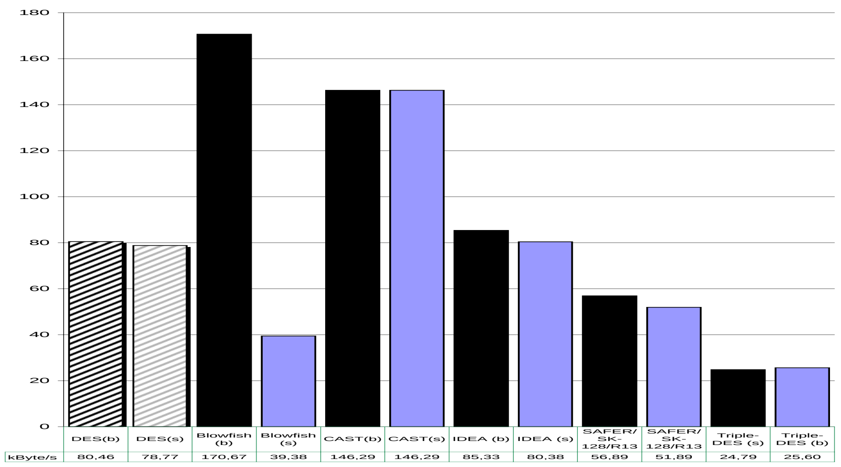
Cipher	DES	Blowfish	CAST	IDEA	SAFER	Triple-DES
1k packet size	80.46	170.29	146.29	85.33	56.89	24.79
100k packet size	78.77	39.36	146.29	80.38	57.89	25.80

Remark:

Our implementation of Triple-DES is not optimized, i.e., both the initial permutation and its inverse are evaluated for each of the three DES steps. We estimate that such trivial modification would speed–up Triple-DES by 10%–20%.

2.7. Conclusions

Blowfish and CAST are both free with fine performance and security. Their memory requirements may be prohibitive for many smartcard implementations though. As a hardware-oriented design, the age-old DES is surprisingly fast. While 56 bits of key size must be considered insecure today, Triple-DES and DESX are reasonably good choices of block ciphers.



	DES(b)	DES(s)	Blowfish (b)	Blowfish (s)	CAST(b)	CAST(s)	IDEA (b)	IDEA (s)	SAFER/ SK-128/R13	SAFER/ SK-128/R13	Triple-DES (s)	Triple-DES (b)
kByte/s	80,46	78,77	170,67	39,38	146,29	146,29	85,33	80,38	56,89	51,89	24,79	25,60

Figure 3. Proven ciphers[46]

3. REMOTELY KEYED ENCRYPTION

Many security relevant applications require storage of cryptographic keys on a tamperproof device, a *smart card.* All key-dependent operations are done on the smart card, such that there is no need to read the key out of the smart card (actually, the smart card should protect the key from being read out). Typically, a smart card is a slow device, compared to an ordinary PC. Hence, key-dependent operations such as en- and decrypting inherently have to be slow, right? Wrong, paradoxically there is still a way of doing fast encryption using a slow smart card.

In 1996, Blaze[6] 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. Blaze defined the "Remotely Keyed Encryption Protocol" (RKEP). In Section 3.1 of his paper, Blaze himself mentions some security problems of his protocols. (Seemingly, these problems where of no importance for the actual application Blaze considered, an encrypting file system for PCs.) More RKEP security problems were pointed out by Lucks,[29] who also gave formal definitions of the security of remotely keyed encryption schemes. These definitions were based on the cryptographic standard definitions of the security of block ciphers.

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. 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 with 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 $\beta \ll B$. (Otherwise, our remotely keyed encryption scheme would not be very efficient.)

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.

Regarding the security of block ciphers and remotely keyed encryption, we consider *two-sided attacks*, 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. (If we concentrate on *chosen plaintext attacks*, we may consider BEAST-RK, the remotely-keyed encryption scheme based on the block cipher BEAST[28]).

We consider the following three security properties:

1. 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
$$(P^{(1)}, C^{(1)}), \ldots, (P^{(q)}, C^{(q)})$$
which are *valid*, i.e. $C_i = \texttt{Encrypt}_K(P_i)$.
2. 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.
3. A remotely keyed encryption scheme is *pseudorandom*, if the block cipher $\texttt{Encrypt}_K : \{0,1\}^B \longrightarrow \{0,1\}^B$ it realizes is pseudorandom.

Note that Blaze's RKEP is neither forgery secure, nor inversion secure, nor pseudorandom, while on the other hand the RaMaRK encryption scheme we present in the next section provably has it all: it is forgery secure and inversion secure and pseudorandom.

Recently, Blaze, Feigenbaum and Naor proposed a remotely keyed encryption scheme which meets even higher security requirements than the ones above.[8]

4. THE RAMARK ENCRYPTION SCHEME

In this section, 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*

$$f : \{0,1\}^b \longrightarrow \{0,1\}^b$$

The proofs of the scheme's security can be found in[29] and assume the building blocks to be secure.

Except when b is tiny, it is completely infeasible to implement truly random functions—one would have to store $b2^b$ bits. In practice, one assumes f to be pseudorandom*. 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. Preenel and Oorschot[34]). Also note that b must be large enough—performing close to $2^{b/2}$ encryptions has to be infeasible. We recommend to choose $b = 160$ or greater. 1 By "$\oplus$" we denote the bit-wise XOR, though mathematically any group operation would do the job as well.

We use three building blocks:

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

1. Key-dependent (pseudo-)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.

   To provide the pseudorandomness, an additional 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 $S(s \oplus t_1)$, $S(s \oplus t_2)$, $\ldots$, 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 $\{0,1\}^b \longrightarrow \{0,1\}^{B-2b}$, tought it is 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 4. In contrast to Blaze's RKEP, B need not be a multiple of b.

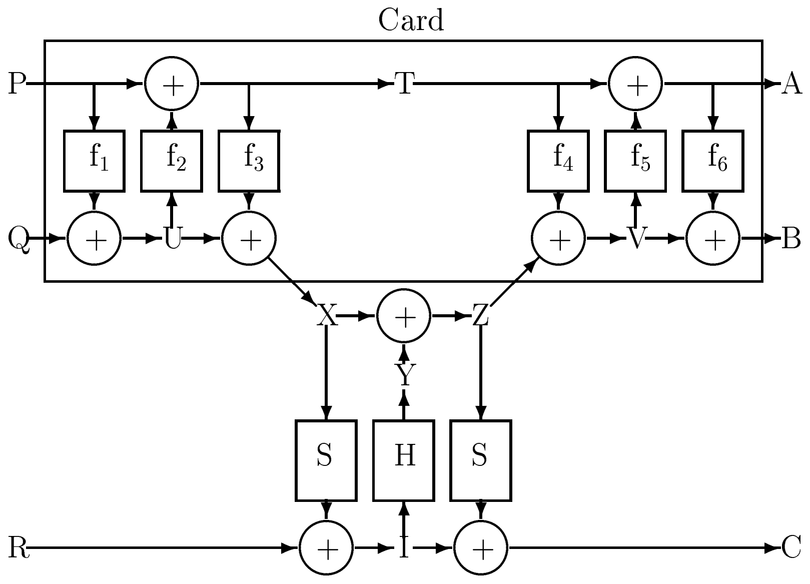


Figure 4. The RaMaRK encryption protocol.

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 $I \in \{0,1\}^{B-2b}$.

Encryption

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.

Decryption

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

1. The host sends A and B to the card.
2. The card computes
$$V = f_6(A) \oplus B \text{ and } T = f_5(V) \oplus A,$$
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.

The RaMaRK scheme is efficient, if the block size B of the cipher it realizes is not too small compared to the security parameter b, because the card itself only operates on $2b$ bit data blocks, while $3b$ bit of information enters the card and the same amount of information leaves the card.

5. BLOCK CIPHERS WITH LARGE BLOCKS

By definition, remotely keyed encryption schemes realize a block cipher. The block cipher realized by the RaMaRK encrypion scheme is called GRIFFIN. Like BEAR, LION, LIONESS[3] and BEAST,[28] GRIFFIN is based on the famous Luby-Rackoff construction.[26] (Actually, BEAR, LION, and BEAST are three-round Luby-Rackoff ciphers, LIONESS is a four-round Luby-Rackoff cipher, and GRIFFIN is composed of three three-round Luby-Rackoff ciphers.) If we use a fast dedicated hash function such as SHA-1[15] or RIPE-MD-160[11] and a fast stream cipher such as SEAL[38] as building blocks (where the hash functions als may be used to realize the pseudorandom functions), all these ciphers are fast and deal well with flexible but large blocks.

Here, "flexible" indicates that the block size may adapt to the data blocks defined by the application. In contrast to ciphers such as BEAR, LION, LIONESS, BEAST, and GRIFFIN, conventional block ciphers such as DES, IDEA and Blowfish work on fixed but small blocks–typically 64 bit. For security reasons, the Luby-Rackoff constructions should not be used to construct block ciphers with such small blocks.

Because of their speed and because of their ability to cope with large but flexibly sized blocks, BEAR, LION, LIONESS, BEAST, and GRIFFIN are interesting for high-speed multimedia applications such as the frame-wise encryption of video data. (The fastest cipher among them is BEAST, which is provably secure against chosen plaintext attacks. Only LIONESS and GRIFFIN are secure against two-sided attacks, and for large blocks we expect GRIFFIN to be significantly faster than LIONESS.)

Remarks on SEAL.

One of the most interesting stream ciphers for our constructions is the *SEAL*.[38] SEAL is a pseudorandom function developed by Phil Rogaway and Don Coppersmith. The encryption of a clear text byte requires approximately five elementary CPU operations. Unlike many other stream ciphers, SEAL allows for random access to the data stream which facilitates synchronization. SEAL is patented by IBM.

After a strong attack by Handschuh and Gilbert[18] , Rogaway and Coppersmith[39] have modified the algorithm which is now available in version 3.0. Hence, we strongly suggest to apply further cryptoanalysis before using SEAL in applications with high security requirements.

6. APPLICATION OF FAST ENCRYPTION IN A DIGITAL LIBRARY

In the TeleTeaching Project at the University of Mannheim[13] lectures are distributed to partner universities over the internet, using MBone[22] technology and our own developments like the MBone Reflector[23] and the digital lecutre board.[16] During a lecture, three data streams are captured at the location of the teacher and transmitted to the remote lecture rooms: video and audio of the teacher and the data produced by a shared whiteboard, which is used to present the lecture material. Furthermore video and audio that is captured in the remote lecture rooms is transmitted between the participating sites to allow questions and enable discussions.

The online lectures can be recorded and stored in our Educational Multimedia Library.[19] To achieve the recording of the online lectures without losing any information, all data streams transmitted between participants (video, audio and whiteboard) must be captured.

As recording whiteboard data needs a specialized recording tool we are actually recording video and audio data and are adding the lecture material in s second step. In our current research, we have developed a prototype of such a recording tool for the digital lecture board.[17] The amount of data needed for a recording depends heavily on parameters like frame or sampling rates, encoding format and compression scheme. For our lectures we are using h.261 with a frame rate of about 5 fps for video encoding and experimented with pcm and gsm for audio encoding and got an average of roughly about 150 MB for a lecture with a duration of 1.30 hours.

Bacher et al.[4] denote, that recording a lecture is a very efficient way of authoring computer-based-training (cbt) material. We are dividing a monolithic recording of a lecture into subparts, which we call learning modules. These learning modules consist of material concerning a specific topic and can be a slide accompanied by video and audio containing the teacher's explanations. The concept of learning modules enables a student to retrieve material on the specific topic he is actually working on. In the Education Multimedia Library a cbt–unit can be created by linking learning modules and enriching the result with additional material like animations or video clips.

Private educational institutions usually want to charge students for taking a specific course. In general, authors normally want to have control over the distribution of their intellectual property and do not allow their material to be given away for free. For this reason, we are planning to integrate a mechanism for accounting and billing into our Educational Multimedia Library. A secure and usable way to control access to a digital library are smartcard systems. As mentioned above, the huge amount of data transferred by a multimedia library can not be decrypted by the card directly.

7. CONCLUSION AND OUTLOOK

We have discussed the performance of block cipher implementations in a machine-independent high-level language such as JAVA. Further, we have presented techniques to "trade away" most of the workload for en- and decryption from a slow smartcard to a fast untrusted host. These techniques are based on the development of Luby-Rackoff ciphers for large blocks. We are still working on a JAVA construction set for Luby–Rackoff ciphers (exspecially Remotely Keyed Systems) and on integrating some of the techniques presented here into the teleteaching software developed at the University of Mannheim.

REFERENCES

1. Adams, C., "Constructing Symmetric Ciphers Using the CAST Design Procedure", in: Designs, Codes and Cryptography, v.12, n. 3, Nov 1997, pp. 71–104.
2. Adams, C., "RFC2144: The CAST-128 Encryption Algorithm", May 1997.
3. Anderson, R., Biham, E., "Two Practical and Provable Secure Blockciphers: BEAR and LION", Proc. of Fast Software Encryption (ed. D. Gollmann), LNCS 1039, Springer, 1996.
4. Bacher, C., Müller, R., Ottmann, T., "Ein Weg zur Integration von Live-Vorlesung, Teleteaching und Lehrsoftwareproduktion Mediengestützte wissenschaftliche Weiterbildung" (in German), Braunschweig, Februar 1997.
5. M. Bellare, M., Rogaway, P., "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols" in: First ACM Conference on Computer and Communications Security, ACM, 1993.
6. Blaze, M., "High-Bandwidth Encryption with Low-Bandwidth Smartcards", in: Fast Software Encryption (ed. D. Gollmann), Springer LNCS 1039, 33–40, 1996.
7. Blaze, M., Diffie, W., Rivest, R., Schneier, B., Shimomura, T., Thompson, E., Wiener, M., "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security", a report by an ad hoc
8. Blaze, M., Feigenbaum, J., and Naor, M., "A Formal Treatment of Remotely Keyed Encryption (Extended Abstract)", Eurocrypt '98, Springer LNCS 1403, 1998, 251-265.
9. Cryptix - Cryptografic Extensions for Java, 1997, `http://www.systemics.com/software/cryptix-java/`
10. RSA-Challange'97, `http://www.rsa.com/des/`
11. 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.
12. Electronic Frontier Foundation, "EFF press release (July 17, 1998): EFF Builds DES Cracker that proves that Data Encryption Standard is insecure", `http://www.eff.org/descracker/`
13. Effelsberg, W.,"Das Projekt TeleTeaching der Universitäten Mannheim und Heidelberg" (in German), Proceedings LEARNTEC'97, Karlsruhe, Germany, Januar 1997.
14. National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard", January 1977.
15. NIST, "Secure Hash Standard", Washington D.C., April 1995.
16. Geyer, W., Weis, R., "A Secure, Accountable, and Collaborative Whiteboard", to appear in: Proc. of IDMS'98, Oslo, Springer LNCS, 1998.
17. Gra, O.: "Realisierung eines Whiteboard-Recorder Moduls", Master's-Thesis, University of Mannheim, Germany, 1998.
18. Handschuh, H., Gilbert, H., "χ^2 cryptoanalysis of the SEAL encryption algorithm". Proc. of Fast Software Encryption 1997, Haifa, Israel, LNCS, Springer, January 1997.
19. Hilt, V., "Educational Multimedia Library Project", `http://www.informatik.uni-mannheim.de/informatik/pi4/projects/emulib/index.en.html`
20. Kilian, J., Rogaway, P., "How to protect DES against exhaustive key search", Proc. of Crypto'96, Advances in Cryptology, Berlin, Springer, 1996.

21. Knudsen, L., "A Key–Schedule Weakness in SAFER K-64", Advances in Cryptology – Crypto '95, LNCS Springer Verlag, 1995, pp. 274–286.
22. Kumar, V., "MBone: Interactive Multimedia on the Internet" , New Riders Publishing, Indianapolis, 1996.
23. Kuhmünch, C.,"Teleteaching over Low-Bandwidth Network Channels", Technical Report TR-98-498, Universität Mannheim, 1998.
24. X. Lai, "Markov ciphers and Differential Cryptoanalyis", Proc. of EUROCRYPT'91, Advances in Cryptology, Springer, 1991.
25. X. Lai, "On the Design and Security of Blockciphers", ETH Series in Information Processing, v. 1, Hartmut-Gorre-Verlag, Konstanz, 1992.
26. Luby, M., Rackoff, C., "How to construct pseudorandom permutations from pseudorandom functions", SIAM J. Computing, Vol 17, No. 2, 1988, pp. 239-255.
27. Lucks, S., "Faster Luby-Rackoff ciphers", Fast Software Encryption (ed. D. Gollmann), LNCS 1039, Springer, 1996.
28. Lucks, S., "BEAST: A fast block cipher for arbitrary blocksize", (ed. Hoprster, P.), Proc. IFIP'96, Conference on Communication and Multimedia Security, Chapman & Hall, 1996, pp. 144–153.
29. Lucks, S., "On the Security of Remotely Keyed Encryption" in: Fast Software Encryption, (ed. E. Biham) Springer LNCS, 1997.
30. Lucks, S., "Attacking Triple Encryption", Fast Software Encryption 5, 1998, (ed. S. Vaudenay), LNCS, Springer, 1998.
31. Lucks, S., "On the Power of Whitening", Manuskript, Universität Mannheim, Fakultät für Mathematik und Informatik.
32. Massey, L.J., "SAFER K-64: A Byte-Orientated Blockciphering Algorithm", Fast Software Encryption, Cambridge Security Workshop Proccedings, LNCS Springer Verlag, 1994, pp. 1–17.
33. Callas, J., Donnerhacke, L., Finnley, H., "OP Formats - OpenPGP Message Format", Internet Draft, November 1997.
34. Preneel, B., van Oorschot, P. "On the Security of Two MAC Algorithms", in: Eurocrypt '96 (ed. U. Maurer), Springer LNCS 1070, 19–32, 1996.
35. Roe, M., "Performance of Symmetric Ciphers and One–way Hash Functions" Fast Software Encryption, Cambridge Security Workshop Proccedings, LNCS Springer Verlag, 1994, pp. 83–86.
36. Roe, M., "Performance of Block Ciphers and Hash Functions – One Year later", Fast Software Encryption, 4th International Workshop Proccedings, LNCS 809, Springer Verlag, 1994, pp. 359–362.
37. Rogaway, P., "The Security of DESX", CryptoBytes, Volume 2, No. 2, RSA Laboratories, Redwood City, CA, USA, Summer 1996.
38. Rogaway, P., Coppersmith, D., "A software-optimized encryption algorithm", Proc. of Cambridge Security Workshop on Fast Software Encryption, (ed. R. Anderson), LNCS 809, Springer, 1994, pp. 56-63.
39. Rogaway, P., Coppersmith, D., "A software-optimized encryption algorithm", revised version Sept. 5, 1997. `http://www.cs.ucdavis.edu/ rogaway/papers/seal.ps`
40. `www.rsa.com/rsa/products/jsafe` or `www.baltimore.ie/jcrypto.htm`, RSA inc. 1998.
41. Schneier, B., "Description of a New Variable-Length Key, 64-Bit Block Cipher", Proc. of Cambridge Security Workshop on Fast Software Encryption, LNCS 809, Springer, 1994, pp. 191-204.
42. Schneier, B., "Applied Cryptography Second Edition", John Wiley & Sons, New York, NY, 1996.
43. Schneier, B., Whiting, D., "Fast Software Encryption: Designing Encryption for Optimal Speed on the Intel Pentium Processor", Fast Software Encryption, 4th International Workshop Proceedings, LNCS Springer Verlag, 1997, pp. 242–259.
44. Simpson, W. A., Baldwin, R., "The ESP DES-XEX3-CBC Transform", Internet-Draft, July 1997.
45. Java Security, 19.11.1997, See: http://www.javasoft.com/security/ 1998.
46. Weis, R., Lucks, S., "The Performance of Modern Block Ciphers in JAVA", CARDIS'98, Louvain-la-Neuve,LNCS , Springer, September 1998.
47. E. Wiewall, Secure Your applications with the Microsoft CryptoAPI, in Microsoft Developer Network News, 5/96,3/4, 1, 1996.