Two Practical and Provably Secure Block Ciphers: BEAR and LION

Ross Anderson[1] and Eli Biham[2]

[1] Cambridge University, England; email rja14@cl.cam.ac.uk
[2] Technion, Haifa, Israel; email biham@cs.technion.ac.il

Abstract. In this paper we suggest two new provably secure block ciphers, called BEAR and LION. They both have large block sizes, and are based on the Luby-Rackoff construction. Their underlying components are a hash function and a stream cipher, and they are provably secure in the sense that attacks which find their keys would yield attacks on one or both of the underlying components. They also have the potential to be much faster than existing block ciphers in many applications.

1 Introduction

There are a number of ways in which cryptographic primitives can be transformed by composition, such as output feedback mode which transforms a block cipher into a stream cipher, and feedforward mode which transforms it into a hash function [P]. A number of these reductions are provable, in the sense that certain kinds of attack on the composite function would yield an attack on the underlying primitive; a recent example is the proof that the ANSI message authentication code is as secure as the underlying block cipher [BKR].

One construction which has been missing so far is a means of building a block cipher out of a stream cipher. In this paper, we show provably secure ways to construct a block cipher from a stream cipher and a hash function. Given that ways of constructing keyed hash functions from stream ciphers already exist [LRW] [K], this enables block ciphers to be constructed from stream ciphers.

On the level of engineering, our constructions allow arbitrary sized blocks to be enciphered in three passes. Given fast software hash functions and stream ciphers, and given that hardware stream ciphers tend to require fewer gates for a given speed than block ciphers do, our constructions may be better than previous schemes in practical applications requiring large block sizes.

Our proposals use DES-like structure. This choice was inspired by the well known paper of Luby and Rackoff, who showed that a good block cipher can be constructed from three good pseudorandom functions by using them as the round functions in a three round Feistel structure [LR]. A simplified treatment of their result can be found in [M1].

The import of their results is that, so long as we believe that (say) $SHA1$ and $SEAL$ [RC] are both good pseudorandom functions, and that their composition $SEAL_{Ki}(SHA1(X))$ is too, we can construct a Luby-Rackoff block cipher as

$$
\begin{aligned}
L &= L\oplus\ SEAL(K1 \oplus SHA1(R)) \\
R &= R\oplus\ SEAL(K2 \oplus SHA1(L)) \\
L &= L\oplus\ SEAL(K3 \oplus SHA1(R))
\end{aligned}
\tag{1}
$$

So long as the keys Ki are independent, this block cipher would be secure. One might hope that it would also be fairly fast, as its computational cost would be three hash functions and three stream ciphers; and as SHA1 runs at about 40 Mbit/sec, and SEAL at over 100 Mbit/sec, we might hope that this cipher would run at about 10 Mbit/sec, which is much faster than most of the block ciphers reported in [R]. The elegance of Luby-Rackoff might lead one to think that this is an optimal construction.

However, as we will now show, we can do substantially better.

2 BEAR

Our first construction, called BEAR, uses two hashes and one stream cipher. Let m be the block size (in bits), which will typically be large (typically of order 1Kbyte–1Mbyte); let $H_K(M)$ be a keyed hash function with the key K for a message M of arbitrary length, and whose result is of a fixed size k bits ($k = 160$ for keyed SHA1 and $k = 128$ for keyed MD5). This keyed hash function could be based on an unkeyed hash function, in which we append and/or prepend the key to the message (a discussion of the relative merits of appending, prepending and both can be found in [T]).

Next, let $S(M)$ be a stream cipher, or more formally a pseudo random function which given the input M will generate an output of arbitrary length (the length used will be clear from the context). Finally, let $|$ denote concatenation of two streams.

BEAR is the unbalanced Feistel cipher [SB] defined as follows. The plaintext P is divided into two parts L and R whose sizes are $|L| = k$ and $|R| = m - k$. The key consists of two (independent) subkeys $K = (K_1, K_2)$, each of which is of length greater than k.

Encryption is done by:

$$
\begin{aligned}
L &= L\oplus\ H_{K_1}(R) \\
R &= R\oplus\ S(L) \\
L &= L\oplus\ H_{K_2}(R)
\end{aligned}
\tag{2}
$$

Decryption is done by:

$$
\begin{aligned}
L &= L\oplus\ H_{K_2}(R) \\
R &= R\oplus\ S(L) \\
L &= L\oplus\ H_{K_1}(R)
\end{aligned}
\tag{3}
$$

We assume that these two functions have the following properties:

1. The keyed hash function $H_K(M)$:
   (a) is based on an unkeyed hash function $H'(M)$, in which we append and/or prepend the key to the message (for example by $H_K(X) = H'(K|M|K)$);
   (b) is one-way and collision-free, i.e. it is hard given Y to find X such that $H'(X) = Y$, and to find unequal X and Y such that $H'(X) = H'(Y)$;
   (c) is pseudo-random, in that even given $H'(X_i)$ for any set of inputs, it is hard to predict any bit of $H'(Y)$ for a new input Y.
2. The stream cipher $S(M)$:
   (a) resists key recovery attacks, in that it is hard to find the seed X given $Y = S(X)$;
   (b) resists expansion attacks, in that it is hard to expand any partial stream of Y.

3 Security of BEAR

Theorem 1. *An oracle which finds the key of BEAR, given one plaintext/ciphertext pair, can efficiently and with high probability find the seed M of the stream cipher S for any particular output $Y = S(M)$.*

Proof. Given $Y = S(M)$ we wish to find M. We generate a plaintext $P = (L, R)$ and a ciphertext $C = (L', R')$ such that $R \oplus R' = Y$, and such that L and L' are random. Since we assume that $|K_1| = |K_2| > k$, with a high probability there are some values of K_1 and K_2 such that $H_{K_1}(R) = L \oplus M$ and such that $H_{K_2}(R) = L' \oplus M$ (although M is still unknown to the attacker). Using the oracle we can find such values of K_1 and K_2. Given K_1 and K_2, we can easily derive M.

Theorem 2. *An oracle which finds the key of BEAR, given one plaintext/ciphertext pair, can efficiently and with high probability find preimages and collisions of the hash function H.*

Proof. Compute the output of the stream cipher for the input 0: $Y_0 = S(0)$. Choose random R, and set $R' = R \oplus Y_0$. Set $L = L' = 0$, and solve the key for plaintext $P = (L, R)$ and the ciphertext $C = (L', R')$. Given this key we actually get two messages, both of which hash to the value zero. By choosing an input other than zero, we can get arbitrary preimages.

These theorems show that a key recovery attack on BEAR would break both the stream cipher and the hash function from which it is constructed. But what about partial attacks?

If can expand the output of the stream cipher (whether by finding the input key or otherwise), then given a ciphertext and a part of the plaintext we can expand our knowledge of the plaintext. On the other hand, if collisions can be found for the hash function, then the attacker might find pairs of plaintexts whose right halves hash to the same value, and thus generate a ciphertext most of whose bits are predictable.

Note that we do not prove any security against attacks which require many blocks to be encrypted under the same key, such as linear and differential attacks. Since in BEAR the block is very large, it may often be possible to arrange things so that each block is encrypted under a different key. However, if differential [BS] or linear [M2] attacks on BEAR are possible, then they can be related to bad properties of either the hash function or the stream cipher (or both).

For example, if after collecting some plaintext-ciphertext pairs we can, with probability greater than one-half, predict bits in the right part of a BEAR ciphertext, then this says that we can predict bits of $S(A \oplus H_K(B))$ where K is unknown and either A or B is new; and if we can predict bits in the left part, then this amounts to a prediction of the exclusive-or of two keyed hashes.

Conversely, if there is a divide-and-conquer attack on the stream cipher [A1], then we can search part of its keyspace by varying the appropriate bits in L. In addition, if we had a hash function with some regularity in its output, so that we could find X_1, X_2 such that $H_K(X_1) \oplus H(X_2)$ lies in a searchable set, then an attacker could find two messages which yielded the same input to the stream cipher in the second round. Such hash function weaknesses were discussed in [A2] and examples found in [V]; the same effect can be obtained even with a good hash function if a user who knows the first key K_1 is in collusion with the attacker. In either case, the attacker can get some known correlation between the plaintexts and the ciphertexts; this will not help her find the keys, but might help her to find some information about future unknown plaintexts. If we wish to make sure that such attacks are not possible, we might add one additional round to BEAR.

4 LION

Our second construction, called LION, is similar except in that we use our stream cipher twice and our hash function only once.

Encryption is done by:

$$
\begin{aligned}
R &= R \oplus S(L \oplus K_1) \\
L &= L \oplus H'(R) \\
R &= R \oplus S(L \oplus K_2)
\end{aligned}
\tag{4}
$$

Decryption is done by:

$$
\begin{aligned}
R &= R \oplus S(L \oplus K_2) \\
L &= L \oplus H'(R) \\
R &= R \oplus S(L \oplus K_1)
\end{aligned}
\tag{5}
$$

In this case, we can have a weaker assumption on the hash function. We do not need to assume that it is a pseudorandom function, but merely that it is collision-free, i.e. that it is hard to find distinct X_1, X_2 such that $H'(X_1) = H'(X_2)$.

The security reduction of LION proceeds similarly to that of BEAR; an oracle which yields the key of LION will break both its components. Partial attacks also reduce, and for example we find that if an attacker knows a preimage of 0 in the hash function, then we get a set of weak keys; wherever $K_1 = K_2$, certain plaintexts will be encrypted to themselves. The details are left as an exercise to the reader.

We note that Kaliski and Robshaw's proposal for basing a block cipher with a large block size on a hash function [KR] would run at about half the speed of the underlying hash function. In addition, as their proposal 'unravels' the hash function into a block cipher, it would have to be evaluated afresh as a new cryptographic primitive. From the raw figures at least, we might expect LION to be the fastest block cipher around, and BEAR to be almost as fast; they would also have the advantage that their security depends on that of already known primitives. So we implemented them to find out.

5 Actual Implementation

We tested both BEAR and LION using blocks of various sizes. For compatibility with Roe's results [R], we used a 133MHz DEC Alpha machine (a Sandpiper server) with SHA1 as the hash function and SEAL as the stream cipher. We keyed SHA1 in BEAR by $H_K(M) = H(K|M|K)$; the stream cipher was $S(M) = SEAL_M(0) — SEAL_M(1) — SEAL_M(2) — \ldots — SEAL_M(\lceil m/(64 \cdot 1024 \cdot 8) \rceil)$. We also used Roe's implementations of both SHA1 and SEAL to ensure strict comparability.

We found that, despite the advertised speeds on a 133MHz Alpha of 41.51 Mbit/sec for SHA1 and 114.8 Mbit/sec for SEAL, we obtained a speed for BEAR of only about 13 Mbit/sec. This varied somewhat according to the block size, from 12.95 Mbit/sec for 64Kb blocks to 13.62 Mbit/sec for 1Mb blocks, suggesting that the key setup times for SEAL were significant. LION confirmed this; instead of being significantly faster, it ran at speeds ranging from 14.83 Mbit/sec for 64Kb blocks up to 18.68 Mbit/sec for 1Mb blocks. Finally, we tested SEAL for various block sizes; we found that when the key setup is included, it runs at 58.7 Mbit/sec with 64Kb blocks, and as little as 7.6 Mbit/sec with 4096 byte blocks — a far cry from the advertised performance.

These results show the need for a stream cipher which runs quickly in software, but does not achieve this at the cost of a very slow key schedule. We are working on candidate stream ciphers which will be the subject of separate publications. Given such ciphers, we expect that BEAR and LION will be highly competitive as fast block ciphers.

6 LIONESS — a four round variant of BEAR and LION

Bear and Lion may not be secure when an adaptive combined chosen plaintext and ciphertext attack is applicable. This attack is similar to the attack on the Luby-Rackoff construction [M1]. In most applications, this attack is not relevant; but for the few cases where it might be, we propose a four-round combined variant of Bear and Lion, called Lioness, which is secure even against this unusual attack.

Lioness uses four independent keys, K_1, K_2, K_3, and K_4. Encryption is done by:

$$\begin{aligned} R &= R \oplus S(L \oplus K_1) \\ L &= L \oplus H_{K_2}(R) \\ R &= R \oplus S(L \oplus K_3) \\ L &= L \oplus H_{K_4}(R) \end{aligned} \tag{6}$$

Decryption is done by:

$$\begin{aligned} L &= L \oplus H_{K_4}(R) \\ R &= R \oplus S(L \oplus K_3) \\ L &= L \oplus H_{K_2}(R) \\ R &= R \oplus S(L \oplus K_1) \end{aligned} \tag{7}$$

7 Modes of Operation

We suggest using the above ciphers with the following modes of operation:

1. Encrypting a single block: the whole message is treated as a single block. This mode has several advantages: (a) Each ciphertext bit depends on all the plaintext bits in a very complex way, which contributes to the cryptographic strength; (b) The security proofs of Bear and Lion hold, since only one block is encrypted with any key; (c) It prevents both differential and linear cryptanalysis, since the attacker cannot get more than one block encrypted under the same key.

2. The message can be divided into several blocks, and each block is encrypted under a different key.
3. Lioness can also be used with the standard modes of DES: Divide the message to several blocks of some length (say 1Kbyte–1Mbyte) and use the standard modes of DES.

Modes of operation whose blocks are long and/or variable may have added attractions in particular application areas. For example, when constructing authentication protocols, the use of a variable length block cipher which treats each protocol message as a single block will prevent splicing attacks of the kind demonstrated by Mao and Boyd on DES-CBC implementations of the Otway-Rees protocol [MB94]. Another example is where the law requires all but 40 bits of the session key to be sent in the clear; if one wishes to get the highest acheivable level of security subject to this constraint, then using a larger block makes the cryptanalyst work harder.

8 Conclusions

Previously it had been known how to construct stream ciphers and hash functions from block ciphers, and hash functions from stream ciphers; by showing how to construct a block cipher from a stream cipher and a hash function, we have completed the set of elementary reductions.

Our constructions possess some interesting provable security properties, and may also be of practical value: they provide fast and strong block ciphers whose block sizes are large and variable. Such ciphers may be useful in a significant number of applications.

Acknowledgment: We would like to thank Mike Roe for making his code available to us for testing, and to the referees for valuable comments.

References

[A1] RJ Anderson, "Solving a Class of Stream Ciphers", in *Cryptologia* v XIV no 3 (July 1990) pp 285–288

[A2] RJ Anderson, "The classification of hash functions", in *Codes and Cyphers — Cryptography and Coding IV* (IMA, 1995) pp 83–93

[BKR] M Bellare, J Kilian, P Rogaway, "The Security of Cipher Block Chaining", in *Advances in Cryptology — CRYPTO 94*, Springer LNCS v 839 pp 341–358

[BS] E Biham, A Shamir, *'Differential Cryptanalysis of the Data Encryption Standard'* (Springer 1993)

[K] H Krawczyk, "LFSR-based Hashing and Authentication", in *Advances in Cryptology — CRYPTO 94*, Springer LNCS v 839 pp 129–139

[KR] BS Kaliski, MR Robshaw, "Fast Block Cipher Proposal", in *Fast Software Encryption*, Springer LNCS 809 (1994) pp 33–40

[LR] Luby, C Rackoff, “How to construct pseudorandom permutations from pseduorandom functions”, in *SIAM Journal on Computing* v 17 no 2 (1988) pp 373–386

[LRW] XJ Lai, RA Rueppel, J Woollven, in *preproceedings of Auscrypt 92* pp 8-7 – 8-11

[M1] U Maurer, “A Simplified and Generalized Treatment of Luby-Rackoff Pseudorandom Permutation Generators’, in *Advances in Cryptology - EUROCRYPT 92*, Springer LNCS v 658 pp 239–255

[M2] M Matsui, “The first experimental cryptanalysis of the Data Encryption Standard”, in *Advances in Cryptology — CRYPTO 94*, Springer LNCS v **839** pp 1–11

[MB94] WB Mao, C Boyd, “Classification of Cryptographic Techniques in Authentication Protocols”, in *Workshop on Selected Areas in Cryptography (SAC 94) — Workshop Record*, pp 95 - 104

[P] B Preneel, *‘Analysis and Design of Cryptographic Hash Functions’*, PhD Thesis, Katholieke Universiteit Leuven, 1993

[R] M Roe, “Algorithms Contest — Preliminary Results”, *preprint handed out at KU Leuven workshop on algorithms*

[RC] P Rogaway, D Coppersmith, “A Software-Optimised Encryption Algorithm”, in *Fast Software Encryption*, Springer LNCS 809 (1994) pp 56–63

[SB] B Schneier, MA Blaze, “McGuffin: an unbalanced Feistel network block cipher”, in *KU Leuven Workshop on Cryptographic Algorithms, preproceedings* p 44

[T] G Tsudik, “Message Authentication with One-Way Hash Functions”, in *Computer Communications Review* v 22 no 5 pp 29 - 38

[V] S Vaudenay, *‘La Sécurité des Primitives Cryptographiques’*, Thèse de Doctorat, Laboratoire d’Informatique de l’Ecole Normale Supérieure, Avril 1995

This article was processed using the LaTeX macro package with LLNCS style