Recent Developments in the Design of Conventional Cryptographic Algorithms

Bart Preneel*, Vincent Rijmen**, and Antoon Bosselaers

Katholieke Universiteit Leuven, Dept. Electrical Engineering–ESAT
Kardinaal Mercierlaan 94, B–3001 Heverlee, Belgium

{bart.preneel,vincent.rijmen,antoon.bosselaers}@esat.kuleuven.ac.be

18 September 1998

Abstract. This paper examines proposals for three cryptographic primitives: block ciphers, stream ciphers, and hash functions. It provides an overview of the design principles of a large number of recent proposals, which includes the global structure, the number of rounds, the way of introducing non-linearity and diffusion, and the key schedule. The software performance of about twenty primitives is compared based on highly optimized implementations for the Pentium. The goal of the paper is to provided a technical perspective on the wide variety of primitives that exist today.

1 Introduction

An increasing number of applications uses software implementations of cryptographic algorithms in order to provide an acceptable security level at a low cost. An important constraint is that the performance of the application should be influenced as little as possible by the introduction of cryptography. The design of secure cryptographic primitives which achieve very high software performance is a challenging problem for the cryptologic research community. This paper intends to report on the state of the art on this problem.

The best which can be achieved currently is to design fast primitives with *some* provable properties, but the general paradigm is still a 'trial-and-error' procedure, which consists of the publication of candidate algorithms and an evaluation by cryptanalysts. In this process, the cryptographic community gathers knowledge on how to design cryptographic algorithms. Note that primitives exist which are provably secure based on some reasonable assumptions (such as the difficulty of factoring the product of two large primes), but these are several order of magnitude slower than the fastest algorithms currently in use.

This paper intends to summarize the state of the art by comparing the different approaches and design choices and their performance in software. It is not

* F.W.O. postdoctoral researcher, sponsored by the Fund for Scientific Research – Flanders (Belgium).

** F.W.O. research assistant, sponsored by the Fund for Scientific Research – Flanders (Belgium).

our intention to give an in-depth security analysis of any primitive or to assess their suitability for a certain application. The main goal is to extract some general principles, such that potential users, cryptanalysts, and designers have an idea of the diverse approaches of the present day cryptographic primitives.

In this paper three types of cryptographic primitives are considered: additive stream ciphers, hash functions, and block ciphers. First these primitives will be introduced briefly. Next, brute force attacks on them will be presented. In §4 the different design principles are described. §5 discusses the evaluation of their security and §6 compares the software performance. The conclusions are given in §7. The status of selected primitives is listed in Appendix A.

2 Cryptographic primitives

This paper focuses on the three most common cryptographic primitives: additive stream ciphers, cryptographic hash functions, and block ciphers. It will be assumed that the reader is familiar with the basic requirements for these primitives, as well as with the ways how these primitives can be used to provide security services such as confidentiality and authentication.

2.1 Additive stream ciphers

Additive stream ciphers stretch a short key and an initial value to a key-stream sequence. If data confidentiality is required, the sender will add this key-stream sequence to the data, simulating the operation of a one-time pad (but without the perfect secrecy). The recipient can recover the data by subtracting the same key-stream sequence. Additive stream ciphers are also known as pseudo-random string generators.

The stream ciphers that are discussed here are 'alleged RC4,' SEAL, and WAKE. The cryptographic literature contains a large number of papers on other constructions derived from linear feedback shift registers (see for example [78]). Important examples include nonlinear filter functions and clock controlled shift registers. They are usually defined at bit level, which makes them more suited for hardware than for software. Although it is certainly possible to adopt these constructions to software environments, they will not be considered in this paper.

2.2 Cryptographic hash functions

Cryptographic hash functions compress strings of arbitrary lengths to strings of fixed lengths (typically 64, 128 or 160 bits). In addition, they satisfy the following properties [54, 65]:

- preimage resistance: it should be hard to find a preimage for a given hash result;
- 2nd preimage resistance: it should be hard to find a 2nd preimage for a given input;

– collision resistance: it should be hard to find two different inputs with the same hash result.

While these properties are simple, experience has learned us that achieving them is quite hard.

Hash functions are used frequently in combination with digital signature schemes; other applications include data integrity, commitment to a string, and the protection of passphrases.

The hash functions discussed in this paper are MD4, MD5, SHA-1, RIPEMD-160, MD2, Snefru, N-hash, Subhash, and Tiger. The first four belong to the so-called 'MD4-family.'

2.3 Block ciphers

Block ciphers transform a relatively short string (typically 64 or 128 bits) to a string of the same length under control of a secret key.

The advantage of block ciphers is that they form a flexible tool that can be used for encryption: the properties of the encryption depend on the mode (ECB, CBC, CFB or OFB-mode) in which the block cipher is used [35, 41]; for example, the OFB-mode provides an additive stream cipher. Block ciphers can also be used to construct other primitives, such as hash functions, and MACs (see §2.4).

The popularity of block ciphers in cryptography is closely related to the popularity of DES, the Data Encryption Standard [29]. The publication of DES as a Federal Information Processing standard in 1977 has influenced conventional cryptography in a major way: DES became widely used to provide cryptographic protection. For some time, the existence of DES (and triple-DES) has made it very difficult for alternative block ciphers to gain acceptance. One can anticipate that this popularity will be continued, as NIST is preparing for a successor of the DES, the AES (Advanced Encryption Standard).

This paper compares the following block ciphers: DES (and 3-DES), FEAL, IDEA, Blowfish, Khufu, SAFER (and variants), LOKI91, CAST, RC5, SHARK, SQUARE, MISTY1 and MISTY2, and 3-WAY.

2.4 Other primitives

The above list of primitives is certainly not complete: Message Authentication Codes (MACs) are conventional cryptographic algorithms which allow a receiver to establish the source and the integrity of data received [65]. Most MAC constructions in use at present are derived from either block ciphers or hash functions. Recently new MACs have been developed in a setting where the secret key is used only once, as for the one-time pad (e.g., MMH by Halevi and Krawczyk [33]). However, these solutions can be made very practical by deriving the actual key from an external key using an additive stream. These new MAC constructions have combinatorial rather than cryptographic properties, which apparently makes it much easier to achieve very high speeds.

Self-synchronizing stream ciphers are important for applications where synchronization is critical, but few dedicated proposals are known. The CFB-mode of a block cipher is a popular way to implement a self-synchronizing stream cipher.

Asymmetric or public-key cryptography plays an important role in many applications; however, to date no asymmetric algorithm has been found which achieves a speed comparable to the primitives discussed above.

Also note that several reductions between primitives have been developed: for example, one can construct a block cipher from an additive stream cipher and a hash function. Two examples of this construction are LION and BEAR [6], both based on a combination of an additive stream cipher and a hash function (SEAL and SHA-1 are used as examples). Reductions can also be used to improve the strength of a given primitive. For example, triple-DES (with two or three keys) and DES-X [39] are DES variants which increase the effective key size. In addition, they can be proved to be at least as secure (or even more secure) than DES itself. While this is certainly important for practical applications (reuse of existing implementations and transfer of trust), it should be pointed out that with the same computational effort, a more secure primitive could be built from scratch. The focus of this paper is mainly on the construction of basic primitives.

3 Brute force attacks

This section reviews brute force attacks on the three primitives.

3.1 Additive stream ciphers

The most important brute force attack is an exhaustive key search. Such an attack requires only a small amount of known plaintext and can be extended to a 'ciphertext only' setting. For an ideal additive stream cipher with k-bit key, searching the key space requires 2^k encryptions. For long term security (10 years or more), k should be at least 75 to 90 bits [9]; 128 bits provide an ample margin.

A potential problem of additive stream ciphers is that the key-stream will eventually repeat. If the internal state consists of m bits, this will happen after about $2^{m/2}$ bits if the next state function is a random function. If the next state function is a permutation, one expects repetitions after 2^{m-1} bits. A sufficiently large value of m can preclude this attack.

In addition, one should be cautious for weaknesses introduced by the resynchronization procedure (see e.g., [23]).

3.2 Hash functions

For a hash function with an n-bit result, brute force attacks to find a preimage or a 2nd preimage require about 2^n hash operations. It should however be noted that if t targets can be attacked simultaneously, the success probability increases with a factor of t (or alternatively, the work factor is reduced with the same

factor). This can be avoided by parameterizing the hash function. For (2nd) preimage resistance, a value of $n = 75$ to 90 is sufficient for 10 years (for a parameterized hash function).

Finding a collision with a square root attack (also known as birthday attack) requires about $2^{n/2}$ operations and very little memory [54, 65]. Moreover, it can be parallelized efficiently. This implies that n should be at least 160 for long-term collision resistance.

3.3 Block ciphers

A first general attack on a block cipher is an exhaustive key search; it requires a few known plaintexts, and can be extended to a 'ciphertext only' attack. As an example, the 56-bit key size of DES poses a serious problem for applications which require more than short term security: with dedicated hardware costing about 1 million US$, a DES key can be recovered in about an hour; the cost per key is less than 50 US$ [92]. Currently several efforts are underway to search for a DES key using idle cycles on the Internet; one expects that the elapsed search time will be less than 5 months.

One can also construct a table which lists all the plaintext/ciphertext pairs of a block cipher for a given key. This attack can be precluded by choosing a large block length, or by changing the key frequently. If the block size n is 64 bits or more, such an attack poses no problem.

Other attacks on block ciphers are based on the way in which the block cipher is used. For example, if one fixed plaintext is encrypted using t different keys, exhaustive key search becomes faster with a factor of t. If an n-bit block cipher is used in CBC, CFB, or OFB mode, information on the plaintext starts to leak after $2^{n/2}$ encryptions [41]. This shows that block lengths of 128 bits are desirable in the near future. It is therefore quite surprising that only two block ciphers (namely RC5 and SQUARE) allow for this block size (SHARK could be extended easily as well). An alternative to a large block size is to change the key frequently, or to use more complex modes (for example, [20, 73]).

An important factor in the development of new block ciphers is the recent progress in cryptanalysis (such as differential [8] and linear cryptanalysis [52]). For DES, these shortcut attacks still require a huge number of chosen respectively known plaintexts, which renders them unrealistic. These new cryptanalytic techniques have brought new insights in the security of cryptographic primitives. Furthermore, these developments have stimulated research on building blocks such as S-boxes and diffusion layers.

4 Design principles for cryptographic algorithms

Designing a slow and secure algorithm is easy for someone who understands the current designs and their strengths and weaknesses (see also §5). For example, composition constructions exist which are at least as secure as each of the building blocks. If on the other hand the performance has to be pushed to

the limits, a thorough analysis has to be combined with a good understanding of the limitations of the processor or technology in which the system has to be implemented. While it is clear that the performance of primitives depends on the implementation details, even for a given environment (processor and memory in software, and power and area constraints in hardware), very little is known on which structures provide the best security (for a given number of instructions per encrypted or hashed byte). Key issues are clearly the use of memory and the amount of parallelism. This section summarizes the different design aspects:

- global structure and number of rounds;
- non-linearity;
- diffusion;
- key schedule.

Note that this selection of topics is tuned towards block ciphers; hash functions do not have a key schedule (but need to specify how to introduce the input bits into each iteration), and the design of additive stream ciphers follows usually a different approach (see § 4.5).

4.1 Global structure and number of rounds

Hash functions and block ciphers operate on relatively large inputs (64 ... 256 bits). These primitives are designed based on a principle proposed by Shannon [83]: nonlinear substitutions are alternated with mixing functions. The result of the combination is that *"any significant statistics from the encryption algorithm must be of a highly involved and very sensitive type—the redundancy has been both* diffused *and* confused *by the mixing transformation."* In the seventies Feistel [28] proposed to use a *round transformation,* consisting of small non-linear components and a transposition (or bit permutation). The strength of a cryptographic primitive can be obtained by repeating this simple transformation. For a block cipher, the secret key has to be introduced in every round as well.

If every input bit is treated in a similar way, one can speak of a *uniform* transformation; sometimes such transformations are called SPN networks (substitution-permutation networks). An example of such a network is given in Figure 1. A disadvantage of this approach is that the inverse function (which is required for decryption in case of a block cipher in ECB or CBC-mode) may be different from the function itself. This can be a problem for hardware and smart card applications. A clear advantage is the inherent parallelism. Examples of block ciphers in this class are SAFER, SHARK, SQUARE, and 3-WAY. Subhash is a hash function using this approach.

A different approach consists of dividing the input into two halves, and applying a non-linear function only to the right half. The result is added into the left half, and subsequently left and right half are swapped. Ciphers following this approach are called Feistel ciphers (see also Figure 2). Since the nonlinear part requires most of the computation, two rounds of a Feistel cipher require about the same effort as a uniform transformation. The output of one nonlinear function is input directly to the next one, which decreases the amount of parallelism

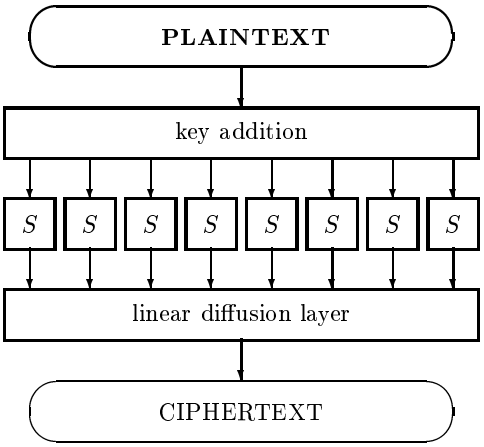


Fig. 1. One round of SHARK, a block cipher with the uniform transformation structure. The nonlinear layer is implemented with eight parallel S-boxes.

but increases the propagation of local changes. Due to the special structure of the round function, the nonlinear function itself need not be invertible, and the round function is an involution. Since DES is a Feistel cipher, more cryptanalytic experience is available on Feistel ciphers than on any other general structure. Other Feistel ciphers are FEAL, Blowfish, Khufu, LOKI91, CAST, and MISTY1.

The approach of a Feistel cipher can be further extended by dividing the input into more parts (ciphers constructed in this way have been called general unbalanced Feistel networks [81]). This may lead to a faster propagation of changes, but could reduce parallelism (depending on the nature of the nonlinear functions). This type of approach is used by the MD4-family, MD2, Tiger, and Snefru.

Other variants and extensions of uniform transformations and Feistel ciphers have been proposed. RC5 is almost a Feistel cipher, since a part of the left half influences the transformation of the right half. MISTY2 is a Feistel variant which increases parallelism: by moving the nonlinear function to a different place, its input in the next round is already known before its output in the current round is calculated, so that calculations for two consecutive rounds can be carried out at the same time. IDEA is a also a variant: a function is computed of the sum of the two halves, and the result is added to the two halves (such that their sum remains a constant).

The main conclusion which can be drawn from this section is that for the time being no conclusion can be drawn on which global structure is best. Different approaches have advantages and disadvantages, and it seems not very likely that clear winners will emerge in the near future.

Number of rounds Most block ciphers and hash functions obtain their strength from repeating a number of identical rounds (one exception is CAST: some

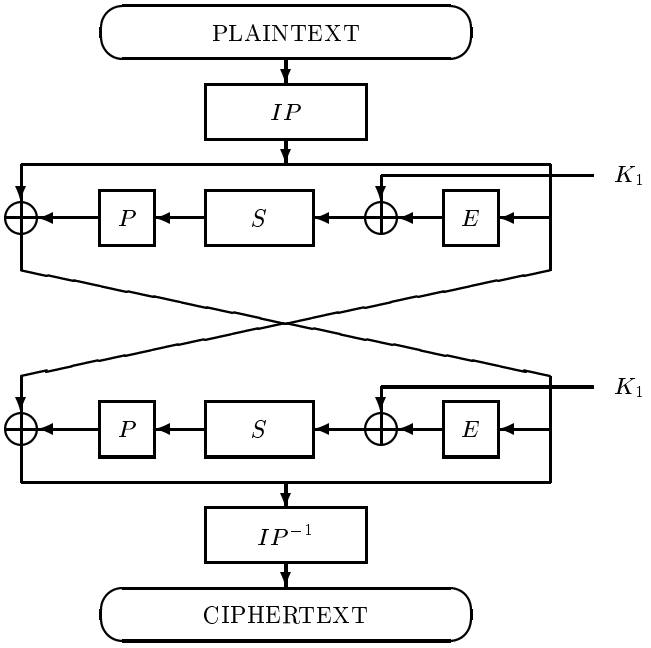


Fig. 2. Two rounds of the DES, the most famous block cipher. It has the Feistel structure. E denotes the linear expansion of the 32 input bits to 48 input bits, $\oplus$ denotes the bitwise exor with the round key, S is the nonlinear substitution and P is the bit permutation.

members of this cipher family have different rounds). In the key paper of Luby and Rackoff [49] it is shown that a 3-round Feistel network can provide a provable secure construction of a pseudo-random permutation from a pseudo-random function (4 rounds are required if both chosen plaintext and chosen ciphertext attacks are allowed). Further extensions of this research have shown that the proofs can be extended if some of the rounds contain functions with combinatorial rather than cryptographic properties [61].

While this provides some theoretical support for the Feistel structure, this work has been misinterpreted by others with regard to its impact on practical ciphers. For example, this research is orthogonal to the issue whether one should have many simple rounds or only three or four more complex rounds. The fact that most proofs seem to achieve their limit at three or four rounds is the consequence of the shortcomings of the model and the proof techniques, rather than a suggestion that ciphers with fewer but more complex rounds are better. An important shortcoming is that while being a pseudo-random permutation is a necessary condition for a good block cipher, it is not sufficient. Moreover, if the round functions are instantiated by a smaller block cipher, other attack models have to be taken into account [45].

There is no doubt one has to check very carefully the resistance to linear and differential attacks. However, one should take into account that several dedicated attacks have been developed which are only applicable to ciphers with a small number of rounds (≤ 8). The most general of these are meet-in-the-middle attacks [17]. Another trick used in these attacks is to peel off one or two rounds (by searching over part of the round keys), and attack the 'weak' structure that remains. Attacks on ciphers with few rounds include:

- meet-in-the middle attacks exploiting the key schedule [17];
- key recovery attacks on 4 or 5 rounds of ladder-DES [45];
- higher order differentials (up to 6–8 rounds);
- interpolation attacks (for ciphers with S-boxes with a simple algebraic structure) [36];
- attacks on Feistel ciphers with non-surjective round functions (up to 8 rounds) [69];
- structure attacks, which exploit the structure of a uniform round transformation [25].

4.2 Nonlinearity

A nonlinear component is essential to every strong cryptographic primitive. The goal of the designer is to build a 'large' nonlinear primitive from smaller ones. The design approaches differ in the choice of the basic nonlinear component.

A straightforward way to implement simple nonlinear functions are lookup tables or S-boxes. The DES uses eight different S-boxes with 6 input bits and 4 output bits (denoted with $6 \rightarrow 4$); the total size of 256 bytes was clearly dictated by hardware constraints of the mid 1970's. Byte level S-boxes ($8 \rightarrow 8$) are very popular as they are suited for software implementations on 8-bit processors.

Ciphers which use such S-boxes include SAFER, MD2, alleged RC4, SHARK and SQUARE. LOKI91 uses S-boxes of type $12 \rightarrow 8$.

For modern processors with 32-bit or 64-bit words, S-boxes with more output bits can provide higher efficiency (although small S-boxes can be implemented efficiently by combining several S-boxes). An important concern is that the S-boxes should fit in the fast cache memory. Snefru was the first cipher to use $8 \rightarrow 32$ S-boxes; this example was followed by Blowfish, Khufu, CAST, and SQUARE. SHARK and Tiger use even larger lookups ($8 \rightarrow 64$). For SHARK and SQUARE these are obtained by combining the byte-level S-boxes with the diffusion operation.

For Blowfish, Khufu and alleged RC4, the S-boxes contain secret values, which makes attacks more difficult: typically, an attacker has to recover all the S-box entries, which is harder than finding the short round keys. In the case of alleged RC4, the value of the S-box is updated during every step.

The value of S-boxes is often selected at random (e.g., MD2 and Snefru), or carefully selected to achieve certain properties (e.g., DES). In both cases built-in trapdoors can be a problem. This has been demonstrated by two of the authors in [68]: this paper shows how an undetectable trapdoor can be built into a block cipher. From the cryptanalytic results available, one can conclude that resistance against linear or differential cryptanalysis is not a property of S-boxes or of linear mappings, but rather of the combination of both. However, for a similar diffusion structure, increasing the size of the S-boxes will increase this resistance (e.g., LOKI91 versus DES).

In the case of SAFER, MISTY1, MISTY2, SHARK, and SQUARE the S-boxes contain some mathematical structure (such as an exponentiation over a finite field). In MISTY1 and MISTY2 the larger S-boxes have been built from smaller Feistel ciphers to allow for an efficient implementation.

If the nonlinear operation does not come from table lookups, it is realized using the processor instructions available. FEAL uses addition and rotation; the MD4-family adds to this also simple bitwise Boolean functions; IDEA uses addition and multiplication. The innovative choice for RC5 was data dependent rotations (in addition to additions and exclusive ors). A special case is 3-WAY: it uses a very simple Boolean function, which can also be considered as a $3 \rightarrow 3$ S-box.

4.3 Diffusion

In order to restrict the complexity of the implementation, nonlinear operations can only be applied to small parts of the block. Several techniques are used to spread local changes.

Linear transformations are very well suited for this purpose. The simplest solution is a bit permutation (or transposition), as is used by DES, LOKI91, and Subhash, or a rotation, as in Khufu and Khafre. An alternative is to add the output of several S-boxes, as is done for Blowfish and CAST. More general linear transformations are the pseudo-Hadamard transform used in SAFER, and the

diffusion operation based on MDS (Maximum Distance Separable) linear codes used in SHARK and SQUARE.

Some cryptographic primitives have no separate diffusion operation, but combine linear and nonlinear operations in such a way that changes are spread quickly through the block. Examples of this approach are FEAL, IDEA, and the MD4-family.

4.4 Key schedule

The key schedule is an important component of a block cipher; it computes the round keys from the external key. For many applications, the key schedule should not be too slow: some applications require very fast key schedules. Examples are constructions for hash functions based on a block cipher, and banking applications which use a new session key per transaction. On the other hand, enumerating the key space should not be too easy in order to frustrate an exhaustive key search. Large key dependent S-boxes (as found in Blowfish and Khufu) do not foster a quick key change.

One issue is the existence of weak keys, i.e., keys for which the block cipher is more vulnerable. Weak keys have been identified for DES, LOKI91, and IDEA. Ideally, such keys should not exist. If they form only a sparse subset of the key space, they pose no security problem if the block cipher is used for encryption; they can be a problem for other applications such as hash functions based on block ciphers.

Recently related key attacks have been developed, in which an attacker obtains ciphertext corresponding to keys with a known or chosen relation (see [38, 41] for more details). The lesson learned from these attacks is that key schedules should not be too simple.

Again many approaches can be distinguished, varying from a selection of bits (DES), over a rotation operation (SAFER, LOKI91, IDEA), to nonlinear operations (CAST). SQUARE uses a key scheduling based on linear codes to guarantee a large distance between round keys from different external keys. The importance of a good key scheduling is demonstrated with the case of SAFER. After the attack of L.R. Knudsen [42] on SAFER-K, the key scheduling has been improved by adding a parity byte to the round keys (this is actually a very simple linear code). The new SAFER is called SAFER-SK.

Some block ciphers such as Blowfish, SHARK, and RC5 use the block cipher itself (with fixed round keys) to perform the key schedule. The hash function SHA-1 uses a variant of a shortened cyclic code to diffuse the message input throughout the calculations (this operation plays the same role as the key schedule of a block cipher). Tiger also applies a diffusion transformation to the message input; it consists of Boolean operations, additions, and rotations.

4.5 Stream ciphers

The above criteria are mostly related towards block cipher and hash functions. Below the most important design choices of three additive stream ciphers are

discussed. The small number of proposals does not yet allow to identify general approaches.

SEAL uses basic operations borrowed from the MD4-family, and is tuned to be fast on the 80486 processor. It derives a large part of its strength from a strong re-keying procedure for every 'block' which uses the hash function SHA-1. This re-keying is quite slow, which implies that the performance depends on the 'block' size.

Alleged RC4 is a very innovative design using a 256-byte table (containing an 8-bit permutation) which is updated during every iteration. Its description is very simple; the basic operations are some additions and table lookups on bytes. The key schedule converts a key of arbitrary size to a random 8-bit permutation.

WAKE uses a series of additions, shifts, and exor operations with the entries of a key dependent table. A variant of WAKE, called WiderWAKE has been developed which can be up to 3.5 times faster by exploiting processor parallelism [18].

5 Security evaluation

As explained in the introduction, the current approach to the design of cryptographic primitives is still largely based on a 'trial-and-error' procedure.

The problems with this approach are that the evaluation phase is typically much more labour intensive than the design phase and provides less guarantees for success. Moreover, the targets chosen are not always the most important ones from a theoretical viewpoint: one will go after the primitive that attracts the most attention (because it is widely used, or because its designer is a well established cryptographer), or that requires least effort. It is safe to state that our knowledge on the design of algorithms has progressed significantly during the last decade, but on the other hand none of the designs currently in use has received what one could call a thorough scrutiny (more than one person-year). This should be compared to the 17 person-year design effort of DES; probably much more effort has been spent on the evaluation of DES.

The lack of evaluation is sometimes compensated for by an 'engineering factor' (over-design with a factor of two), which is acceptable for applications which can afford it, but poses problems for critical applications such as hard disk encryption. This is of course not an excuse to use unpublished algorithms which achieve a high performance at the cost of security (adding the same 32-bit key to all the data words is a widespread example of this). On the other hand, it is quite clear that someone with a good understanding of present day cryptanalysis can design a secure but slow algorithm with a very limited effort:

> For a block cipher, it is sufficient to define a round function based on a nonlinear operation (avoid linear relations) and a simple mixing component (to spread local changes); add round keys in between the rounds (and at the beginning and end of the cipher), which are derived in a complex way from the key (e.g., by using the block cipher itself with fixed round keys). If the number of rounds is 32, or even better 64, breaking

this slow block cipher will be very difficult. (Of course it is possible to follow this 'recipe' and to come up with a weak cipher, but this will require some cryptographic skills !)

If a cipher is a variant or extension of an existing construction (such as DES with modified S-boxes and key schedule), a quick evaluation based on known cryptanalytic techniques can provide a good insight in the basic security. If these modifications have been made to avoid certain attacks, there exists however the possibility that improved versions of these attacks will still apply; this can lead to a loss of the intended improvement. The more innovative a cipher is, the more work needs to be done on the evaluation, since cryptanalysis requires the development of completely new techniques.

Recently some progress has been made with respect to the existing paradigm by ciphers which were proven to be secure against differential and/or linear cryptanalysis. The first construction was the Nyberg-Knudsen scheme [62], followed by constructions by Nyberg [63], and MISTY1 and MISTY2 by Matsui [53]. One should however keep in mind that 'provable security' is probably one of the most misused terms in cryptography and that provable security against one or two important attacks does not imply that the cipher is secure: other attacks might exist, such as attacks based on truncated differentials (Knudsen, [43]). This can be illustrated by a variant of the Nyberg-Knudsen scheme, which turns out the be very vulnerable to an interpolation attack [36]. On the other hand, provable security against certain attacks is certainly a first step in the right direction.

6 Performance evaluation

Optimizing the performance of software and hardware is quite different. Fast hardware relies on parallelism and pipelining, while for software the access to memory is a key to high performance: the designer tries to minimize access to slow memory, and to stay as much "on chip" (registers and cache memory) as possible. However, parallelism becomes more important for software as well: recent evolutions in general purpose processors are clearly oriented towards more inherent parallelism, both on the hardware level (multiple execution units) and the software level (SIMD instructions). Two basic multiple-issue architectures can be distinguished: superscalar and very long instruction word (VLIW).

- A *superscalar* processor has dynamic issue capability: a varying number of instructions is issued every clock cycle. The hardware dynamically decides which instructions are simultaneously issued and to which execution units, based on issue criteria and possible data dependencies.
- A *VLIW* processor has fixed issue capability: every clock cycle a fixed number of instructions is issued, formatted as one long instruction (hence the name). The software (i.e., the compiler) is completely responsible for creating a package of instructions that can be issued simultaneously. No decisions about multiple issue are dynamically taken by the hardware.

An example of the speedup that can be achieved by this kind of parallel processing is given by WiderWAKE, which runs at 50 MByte/s on a 100 MHz Philips TriMedia processor, a 32-bit VLIW processor capable of executing up to 5 operations in parallel [85]. This is more than three times faster than WAKE in OFB mode. It is a challenge for the designer of new cryptographic primitives to exploit this parallelism in an optimal way, without compromising security.

An additional way to exploit parallelism inherent in many algorithms is single-instruction, multiple-data (SIMD) processing. A SIMD instruction performs the same operation in parallel on multiple data elements, packed into a single processor word. Tuned to accelerate multimedia and communications software, these instructions can be found in an increasing number of general-purpose processor architectures. Examples are Intel's MMX [64], UltraSPARC's VIS [86], PA-RISC 2.0 architecture's MAX [47], Alpha's MVI [77], and MIPS's MDMX [58]. MMH [33] is an example of a MAC taking advantage of this newly emerging technology (cf. §2.4).

The problem of accessing slow memory becomes more and more important as the memory access time seems to decrease more slowly than the cycle time of the processors. This suggests that faster cryptographic primitives will make use of logic and arithmetic operations available on a standard processor and of relatively small S-boxes, i.e., typically a few Kbyte in order to fit in the primary, on-chip cache. Opening up new perspectives is the recent trend to include ever larger secondary caches on a dedicated bus limiting latency to a minimum. The advantage of S-boxes is that they can yield a strong nonlinear relation between input and output. S-boxes with 8 input bits and 32 or 64 output bits seem to be particularly suited for the current 32-bit or 64-bit architectures.

Other, less important aspects which influence software performance are word size, byte ordering ("endianness" [19]), and carries, which tend to be processor dependent. Running an algorithm on a processor that has a smaller word size than that of the algorithm will normally result in reduced performance, as it requires more instructions to do the same operations. On a processor having a larger word size than that of the algorithm advantage might be taken of the new SIMD instructions, if available. Byte ordering influences performance if endian-sensitive operations are used (like add, multiply, and rotate over non-multiples of 8) and the processor doesn't support the kind of addressing mode the algorithm requires. In that case an explicit conversion between little and big endian order is needed on all input and output data. Moreover, this overhead becomes relatively more important as the algorithm becomes faster. However, the use of endian-neutral operations (like Boolean operations), possibly in combination with table lookups, allows for the algorithm description to be adapted to the byte ordering convention of the particular processor the algorithm is run on. This involves e.g., redefining the lookup tables and adapting the selection of address bits for lookup purposes. In that case the algorithm performance is independent of the byte order. All algorithms either use endian-neutral operations or specify the endian-convention to be employed, except for SEAL. SEAL uses endian-sensitive operations and outputs 32-bit words, but its description [75] does not specify a

particular endian convention. Instead it is suggested to allow either convention and to include in SEAL-encrypted ciphertext information indicating the endian convention used.

No such thing as "*the* software performance" of a given algorithm exists, even if figures are given for a specific processor. The key is again the use of memory: very different results are obtained depending on whether the data resides in (level 1 or level 2) cache memory, in main memory, or on disk. On the other hand, one wants to measure the performance of the algorithm rather than of the computer. Other factors influencing the performance of an algorithm's implementation include:

- equivalent representations and the use of tables: this can yield significant speed-ups, mainly for algorithms which are not designed specifically for software.
- quality of the compiler: for high level languages, good compilers (with the right optimizations activated) can produce code which is almost an order of magnitude faster; hand coded assembly language sometimes results in significant improvements by using processor instructions such as rotate which are not implemented in languages such as C, by optimizing the use of registers, and by exploiting more efficiently instruction-level parallelism.

One can ask the question whether one should try to optimize the design towards a single processor: designing and reviewing a cryptographic algorithm will take several years, and by that time the processor will probably be outdated. But, most processors are downward compatible, and if one tunes an algorithm to a recent processor without exploiting particular features (such as the availability of certain carries), it is very likely to achieve a high speed on most other processors as well (SEAL is a good example, cf. infra). Finally, it should be noted that the evolution of processors on smart cards is significantly slower than that of general purpose processors. Therefore, there is still an interest in new algorithms for 'old' processors.

Table 1 gives an overview of the speed of the most important algorithms. In order to measure as much as possible the performance of the algorithm (and not of the compiler or of the computer used), and in order to compare the algorithms on an equal basis, it was decided to implement all the algorithms on a single, widely used processor, a (90 MHz) Pentium. The most important features of the Pentium are a complex instruction set, a 32-bit word size, a small register set of 7 general-purpose registers, little-endian addressing, a 2-way superscalar architecture, and separate on-chip code and data caches of 8K each. All implementations have been written in assembly language, and all of them have been optimized to the same (high) level. A comparison of the CISC (Complex Instruction Set Computer) Intel Architecture with the most popular RISC (Reduced Instruction Set Computer) architectures (MIPS IV, PA-RISC 2.0, PowerPC, SPARC V9, Alpha EV5) learns us that the Pentium can to a certain extent be considered as a lower bound: current RISC features include 64-bit word size, at least 31 general-purpose registers, provisions for both little and big-endian addressing, and up to 32K of primary (data) cache. Algorithms

Table 1. Performance in clock cycles per block of output and Mbit/s of several additive stream ciphers, hash functions, and block ciphers on a 90 MHz Pentium. All implementations are written in assembly language, and their level of optimization is comparable. Code and data are assumed to reside in the on-chip caches (except for Snefru, Tiger, and SHARK, that require more data memory than the 8K of the primary data cache). Only the required memory for tables is listed. Some algorithms require additional memory for storing the state and possibly the round keys.

Algorithm	Rounds or (Steps)	Block (bits)	Word (bits)	Endian-ness	Tables (bytes)	Performance (cyc./blk)	(Mbit/s)
(alleged) RC4		8192	8	n.a.	256	6711	110
SEAL 3.0		8192	32	select.	3K+16/1K	3727[a]	198[a]
MD2	18	128	8	n.a.	256	2709	4.25
MD4	(48)	512	32	little	none	241	190.6
MD5	(64)	512	32	little	none	337	136.2
RIPEMD-128	(2×64)	512	32	little	none	592	77.6
RIPEMD-160	(2×80)	512	32	little	none	1013	45.3
SHA-1	$(80 + 64)$	512	32	big	none	837	54.9
Snefru-128	8	384	32	neutral	16K	5730[b]	6.03
Snefru-256	8	256	32	neutral	16K	5738[b]	4.02
Tiger	24+2	512	64	little	8K	1320[c]	34.9
Blowfish	16	64	32	big	4K	158	36.5
CAST	16	64	32	big	4K	220	26.2
DES	16	64	32	neutral	2K	340	16.9
3DES	48	64	32	neutral	2K	928	6.20
IDEA	8.5	64	16	big	none	590[d]	9.75
Khufu	32	64	32	neutral	4K	132	43.6
RC5-32/12	12	64	32	little	none	151[e]	38.1
RC5-32/16	16	64	32	little	none	199[e]	28.9
SAFER K-64	6	64	8	n.a.	512	258	22.3
SAFER SK-64	8	64	8	n.a.	512	338	17.0
SAFER (S)K-128	10	64	8	n.a.	512	418	13.8
SHARK	6	64	64	neutral	16K	585[f]	9.85
RC5-64/24	24	128	64	little	none	830[g]	13.9
SQUARE	8	128	32	neutral	4K	244	47.2

[a] Figures are for a little endian convention. Big endian overhead amounts to 384 cycles per 1024 byte block, resulting in a reduced throughput of 179 Mbit/s.

[b] The tables are twice as large as the on-chip cache. If all used data were cached, Snefru's performance would more than double (to 2674 and 2682 cycles, respectively).

[c] The tables are equally large as the on-chip cache. Additional memory is required for the state (48 bytes), a copy of the data (64 bytes), and the data buffer. Completely cached data would improve performance by nearly 30% (1033 cycles, or 44.6 Mbit/s).

[d] Performance suffers from the slow integer multiplication taking 9 cycles. A 1-cycle multiply would almost double the speed of IDEA (to 318 cycles).

[e] For RC5-32/r: $7 + 12r$ cycles per input block.

[f] The tables are twice as large as the on-chip cache. If all used data were cached, performance would improve with more than a factor 4 (133 cycles, or 43.2 Mbit/s).

[g] For RC5-64/r: $14 + 34r$ cycles per input block.

doing well on the Intel Architecture are expected to do well on any modern, 32-bit processor [74]. An important exception are algorithms using a rotate instruction, which is not available on all RISC architectures.

Most algorithms, despite the fact that their operations are basically serial in nature, contain enough instruction-level parallelism to keep both execution units of the Pentium busy for most of the time. Some algorithms, e.g., SHA-1 and RIPEMD-160, contain even more parallelism than most current general-purpose processors are able to provide [13]. More implementation details concerning the MD4-family of hash functions can be found in [11, 14]. Only Khufu contains hardly any instruction-level parallelism, so that its performance does not benefit much from having parallel execution units. Also the inherent parallelism of SEAL is limited, but it is nevertheless remarkably fast on the Pentium, which is explained by the fact that it was tuned by its designers to be fast on 80x86 processors (use of registers, choice and scheduling of operations). In response to the attack of [34] two additional exors for each 16 bytes of output have been added making SEAL 3.0 [75] about 3.6% slower than the original SEAL 1.0 [74]. Algorithms with word sizes larger than 32 bits (e.g., SHARK, RC5-64/24, and Tiger) will perform relatively better on 64-bit architectures. The SHARK implementation has the additional disadvantage on a Pentium of having tables that are too large to fit into the on-chip data cache, a property it shares with Snefru. Tiger has tables equally large as the on-chip cache, but this is still a problem, as additional memory is required for the state, a copy of the input data, and the data buffer itself. IDEA suffers from the Pentium's slow integer multiplication (9 cycles), but a 1-cycle multiply would still only result in a doubling of the speed, highlighting the fact that IDEA is more than just integer multiplications. An interesting alternative is an IDEA implementation using the SIMD MMX-instructions, providing both a fast and parallel (but unfortunately only signed) multiplication. Such an implementation requires in the order of 360 cycles per 64-bit block, being about 1.6 times faster than a Pentium only implementation [48]. Some algorithms (e.g., Snefru and SHA-1) will perform relatively better on processors having a large register set, such as most RISC processors: this enables the complete internal state to be kept in registers. Most figures of Table 1 confirm or improve upon those mentioned in [82], except for IDEA, where the authors confirmed that their figure of 400 cycles should only be considered as a very cursory lower bound, that has not been substantiated by an implementation [91].

Running these implementations on a PentiumPro or a PentiumII will not necessarily result in the same relative speeds: some implementations heavily rely on reading a 32-bit register shortly after having written to an 8-bit subset of the same register. On the successors of the Pentium this results in so-called partial register stalls, each of which accounts for at least 7 cycles, hence degrading performance considerably.

7 Conclusions

At present, it is not possible to design a additive stream cipher, hash function, or block cipher which is both very fast and 'secure'. This can be summarized in the following quotes:

> L. O'Connor: *"Most ciphers are secure after sufficiently many rounds."*
> J.L. Massey: *"Most ciphers are too slow after sufficiently many rounds."*

Some progress has been made into the direction of provable security, often at the cost of performance. What does exist however, is provable insecurity, i.e., for some designs, serious weaknesses have been identified.

Given the fact that most fast designs are still developed in a 'trial-and-error paradigm' and that very little evaluation effort is available, the reader is cautioned against adopting new cryptographic primitives too quickly. While the cryptographic community has made significant progress during the last twenty years, our knowledge is still very limited; existing cryptanalytic results should be evaluated with great care, even if they are only of theoretical nature.

References

1. C.M. Adams, "Simple and effective key scheduling for symmetric ciphers," *Proceedings of SAC'94, Workshop on Selected Areas in Cryptography*, pp. 129–133.
2. C.M. Adams, "Constructing symmetric ciphers using the CAST design procedure," *Designs, Codes, and Cryptography*, Vol. 12, No. 3, November 1997, pp. 71–104.
3. C.M. Adams, S.E. Tavares, "The structured design of cryptographically good S-boxes," *Journal of Cryptology*, Vol. 3, No. 1, 1990, pp. 27–42.
4. C.M. Adams, S.E. Tavares, "Designing S-boxes for ciphers resistant to differential cryptanalysis," *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography*, W. Wolfowicz, Ed., Fondazione Ugo Bordoni, 1993, pp. 181–190.
5. R. Anderson, E. Biham, "Tiger: a fast new hash function," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 89–97.
6. R. Anderson, E. Biham, "Two practical and provably secure block ciphers: BEAR and LION," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 113–120.
7. K. Aoki, K. Ohta, "Differential-linear cryptanalysis of FEAL-8," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences,* Vol. E79-A, No. 1, January 1996.
8. E. Biham, A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard,* Springer-Verlag, 1993.
9. M. Blaze, W. Diffie, R.L. Rivest, B. Schneier, T. Shimomura, E. Thompson, M. Wiener, "Minimal key lengths for symmetric ciphers to provide adequate commercial security," January 1996.
10. J. Borst, L.R. Knudsen, V. Rijmen, "Two attacks on reduced IDEA," *Advances in Cryptology, Proc. Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, pp. 1–13.

11. A. Bosselaers, R. Govaerts, J. Vandewalle, "Fast hashing on the Pentium," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 298–312.
12. A. Bosselaers, H. Dobbertin, B. Preneel, "The RIPEMD-160 cryptographic hash function," *Dr. Dobb's Journal*, Vol. 22, No. 1, January 1997, pp. 24–28.
13. A. Bosselaers, R. Govaerts, J. Vandewalle, "SHA: a design for parallel architectures?," *Advances in Cryptology, Proc. Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, pp. 348–362.
14. A. Bosselaers, "Even faster hashing on the Pentium," Presented at the rump session of Eurocrypt'97, Konstanz, Germany, May 12–15, 1997, and updated on November 13, 1997. Available from `ftp://www.esat.kuleuven.ac.be/pub/COSIC/bosselae/pentiumplus.ps.gz`.
15. L. Brown, M. Kwan, J. Pieprzyk, J. Seberry, " Improving resistance to differential cryptanalysis and the redesign of LOKI," *Advances in Cryptology, Proceedings Asiacrypt'91, LNCS 739*, H. Imai, R.L. Rivest, and T. Matsumoto, Eds., Springer-Verlag, 1993, pp. 36–50.
16. C. Charnes, L. O'Connor, J. Pieprzyk, R. Safavi-Naini, and Y. Zheng, "Comments on Soviet encryption algorithm," *Advances in Cryptology, Proceedings Eurocrypt'94, LNCS 950*, A. De Santis, Ed., Springer-Verlag, 1995, pp. 433–438.
17. D. Chaum, J.-H. Evertse, "Cryptanalysis of DES with a reduced number of rounds — sequences of linear factors in block ciphers," *Advances in Cryptology, Proceedings Crypto'85, LNCS 218*, H.C. Williams, Ed., Springer-Verlag, 1985, pp. 192–211.
18. C.S.K. Clapp, "Optimizing a fast stream cipher for VLIW, SIMD, and superscalar processors," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 273–287.
19. D. Cohen, "On holy wars and a plea for peace," *IEEE Computer*, Vol. 14, No. 10, October 1981, pp. 49–54.
20. D. Coppersmith, D.B. Johnson, S.M. Matyas, "A proposed mode for triple-DES encryption," *IBM Journal of Research & Development*, Vol. 40, 1996, pp. 253–261.
21. T.W. Cusick, M.C. Wood, "The REDOC-II cryptosystem," *Advances in Cryptology, Proceedings Crypto'90, LNCS 537*, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 545–563.
22. J. Daemen, "Cipher and Hash Function Design. Strategies Based on Linear and Differential Cryptanalysis," *Doctoral Dissertation*, Katholieke Universiteit Leuven, 1995.
23. J. Daemen, R. Govaerts, J. Vandewalle, "Resynchronization weaknesses in synchronous stream ciphers," *Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 159–169.
24. J. Daemen, R. Govaerts, J. Vandewalle, "A new approach to block cipher design," *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 18–32.
25. J. Daemen, L.R. Knudsen, V. Rijmen, "The block cipher SQUARE," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 149–165.
26. J. Daemen, L.R. Knudsen, V. Rijmen, "The block cipher SQUARE algorithm," *Dr. Dobb's Journal*, Vol. 22, No. 10, October 1997, pp. 54–57.
27. H. Dobbertin, A. Bosselaers, B. Preneel, "RIPEMD-160, a strengthened version of RIPEMD," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 71–82.
28. H. Feistel, "Cryptography and computer privacy," *Scientific American*, Vol. 228, No. 5, May 1973, pp. 15–23.

29. FIPS 46, *"Data Encryption Standard,"* Federal Information Processing Standard (FIPS), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January 1977.
30. FIPS 180-1, *"Secure Hash Standard,"* Federal Information Processing Standard (FIPS), Publication 180-1, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., April 1995.
31. H. Gilbert, P. Chauvaud, "A chosen plaintext attack of the 16-round Khufu cryptosystem," *Advances in Cryptology, Proceedings Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 359–368.
32. J. Golić, "Linear statistical weakness of alleged RC4 keystream generator," *Advances in Cryptology, Proc. Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, pp. 226–238.
33. S. Halevi, H. Krawczyk, "MMH: Software message authentication in the Gbit/second rates," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 172–189.
34. H. Handschuh, H. Gilbert, "χ^2 Cryptanalysis of the SEAL encryption algorithm," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 1–12.
35. ISO/IEC 10116, *"Information technology – Security techniques – Modes of operation of an n-bit block cipher algorithm,"* IS 10116, 1991.
36. T. Jakobsen, L. Knudsen, "The interpolation attack on block ciphers," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 28–40.
37. B.S. Kaliski, "The MD2 Message-Digest algorithm," *Request for Comments (RFC) 1319*, Internet Activities Board, Internet Privacy Task Force, April 1992.
38. J. Kelsey, B. Schneier, D. Wagner, "Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 237–252.
39. J. Kilian, P. Rogaway, "How to protect DES against exhaustive key search," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 252–267.
40. L.R. Knudsen, "Block ciphers – analysis, design and applications," *PhD. Thesis, DAIMI PB 485*, Aarhus University, 1994.
41. L.R. Knudsen, "Block ciphers – a survey," *This Volume.*
42. L.R. Knudsen, "A key-schedule weakness in SAFER-K64," *Advances in Cryptology, Proceedings Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 274–286.
43. L.R. Knudsen, T.A. Berson, "Truncated differentials of SAFER," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 15–26.
44. L.R. Knudsen, W. Meier, "Improved differential attack on RC5," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 216–228.
45. L.R. Knudsen, "On the security of Bear & Lion & ladder-DES," Presented at the rump session of the Fast Software Encryption Workshop, Haifa, Israel, January 20–22, 1997.
46. X. Lai, J.L. Massey, S. Murphy, "Markov ciphers and differential cryptanalysis," *Advances in Cryptology, Proceedings Eurocrypt'91, LNCS 547*, D.W. Davies, Ed., Springer-Verlag, 1991, pp. 17–38.
47. R. Lee, "Subword parallelism with MAX-2," *IEEE Micro,* Vol. 16, No. 4, August 1996, pp. 51–59.
48. H. Lipmaa, "IDEA: A cipher for multimedia architectures?," *Selected Areas in Cryptography, LNCS*, Springer-Verlag, 1999.

49. M. Luby, C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions," *SIAM Journal on Computing*, Vol 17, No. 2, April 1988, pp. 373–386.
50. J.L. Massey, "SAFER-K64: A byte oriented block-ciphering algorithm," *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 1–17.
51. J.L. Massey, "SAFER K-64: One year later," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 212–241.
52. M. Matsui, "Linear cryptanalysis method for DES cipher," *Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765*, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 386–397.
53. M. Matsui, "New block encryption algorithm MISTY," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 54–68.
54. A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
55. R.C. Merkle, *Secrecy, Authentication, and Public Key Systems,* UMI Research Press, Ann Arbor, Michigan, 1979.
56. R. Merkle, "Fast software encryption functions," *Advances in Cryptology, Proceedings Crypto'90, LNCS 537*, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 476–501.
57. R. Merkle, "A fast software one-way hash function," *Journal of Cryptology*, Vol. 3, No. 1, 1990, pp. 43–58.
58. "MIPS extension for digital media with 3D," MIPS Technologies, Inc., March 12, 1997.
59. S. Miyaguchi, "The FEAL cipher family," *Advances in Cryptology, Proceedings Crypto'90, LNCS 537*, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 627–638.
60. S. Moriai, K. Aoki, K. Ohta, "The best linear expression search of FEAL," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences,* Vol. E79-A, No. 1, January 1996.
61. M. Naor, O. Reingold, "On the construction of pseudo-random permutations: Luby-Rackoff revisited," *Security in Communication Networks*, Amalfi (I), September 16–17, 1996.
62. K. Nyberg, L.R. Knudsen, "Provable security against a differential attack," *Journal of Cryptology*, Vol. 8, No. 1, 1995, pp. 27–38.
63. K. Nyberg, "Generalized Feistel networks," *Advances in Cryptology, Proceedings Asiacrypt'96, LNCS 1163*, K. Kim and T. Matsumoto, Eds., Springer-Verlag, 1996, pp. 91–104.
64. A. Peleg, U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro,* Vol. 16, No. 4, August 1996, pp. 42–50.
65. B. Preneel, "Hash functions and MACs: state of the art," *This Volume.*
66. V. Rijmen, "Cryptanalysis and design of iterated block ciphers," *Doctoral Dissertation*, Katholieke Universiteit Leuven, 1997.
67. V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, E. De Win, "The cipher SHARK," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 99–111.
68. V. Rijmen, B. Preneel, "A family of trapdoor ciphers," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 139–148.
69. V. Rijmen, B. Preneel, E. De Win, "On weaknesses of non-surjective round functions," *Designs, Codes, and Cryptography*, Vol. 12, No. 3, November 1997, pp. 251–264.
70. R.L. Rivest, "The MD4 message-digest algorithm," *Request for Comments (RFC) 1320*, Internet Activities Board, Internet Privacy Task Force, April 1992.

71. R.L. Rivest, "The MD5 message-digest algorithm," *Request for Comments (RFC) 1321*, Internet Activities Board, Internet Privacy Task Force, April 1992.
72. R.L. Rivest, "The RC5 encryption algorithm," *Fast Software Encryption, LNCS 1008*, B. Preneel, Ed., Springer-Verlag, 1995, pp. 86–96.
73. R.L. Rivest, "All-or-nothing encryption and the package transform," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 210–218.
74. Ph. Rogaway, D. Coppersmith, "A software-optimized encryption algorithm," *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 56–63.
75. Ph. Rogaway, D. Coppersmith, "A software-optimized encryption algorithm," *Journal of Cryptology*, to appear. Available from `http://www.cs.ucdavis.edu/~rogaway/papers/seal.ps`.
76. A. Roos, "A class of weak keys in the RC4 stream cipher," *preliminary draft*, 1996.
77. P. Rubinfeld, B. Rose, M. McCallig, "Motion Video Instruction Extensions for Alpha," Digital Equipment Corporation, October 18, 1996.
78. R.A. Rueppel, "Stream ciphers," in *"Contemporary Cryptology: The Science of Information Integrity,"* G.J. Simmons, Ed., IEEE Press, 1991, pp. 65–134.
79. K. Sakurai, S. Furuya, "Improving linear cryptanalysis of LOKI91 by probabilistic counting method," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 114–133.
80. B. Schneier, "Description of a new variable-length key, 64-bit block cipher (Blowfish)," *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 191–204.
81. B. Schneier, J. Kelsey, "Unbalanced Feistel networks and block cipher design," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 121–144.
82. B. Schneier, D. Whiting, "Fast software encryption: designing encryption algorithms for optimal software speed on the Intel Pentium processor," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 242–259.
83. C.E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, Vol. 28, No. 4, 1949, pp. 656–715.
84. A. Shimizu, S. Miyaguchi, "Fast data encipherment algorithm FEAL," *Advances in Cryptology, Proceedings Eurocrypt'87, LNCS 304*, D. Chaum and W.L. Price, Eds., Springer-Verlag, 1988, pp. 267–278.
85. G.A. Slavenburg, S. Rathnam, H. Dijkstra, "The Trimedia TM-1 PCI VLIW media processor," *Hot Chips VIII Conference*, Stanford University, Palo Alto, CA, 1996.
86. M. Tremblay, J.M. O'Connor, V. Narayanan, L. He, "VIS speeds new media processing," *IEEE Micro,* Vol. 16, No. 4, August 1996, pp. 10–20.
87. W. Tuchman, "Hellman presents no shortcut solutions to DES," *IEEE Spectrum*, Vol. 16, No. 7, July 1979, pp. 40–41.
88. P.C. van Oorschot, M. Wiener, "A known-plaintext attack on two-key triple encryption," *Advances in Cryptology, Proceedings Eurocrypt'90, LNCS 473*, I.B. Damgård, Ed., Springer-Verlag, 1991, pp. 318–325.
89. S. Vaudenay, "On the weak keys of Blowfish," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 27–32.
90. D.J. Wheeler, "A bulk data encryption algorithm," *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 127–134.
91. D. Whiting, personal communication, May 1997.
92. M.J. Wiener, "Efficient DES key search," *Technical Report TR-244*, School of Computer Science, Carleton University, Ottawa, Canada, May 1994. Presented at the

rump session of Crypto'93 and reprinted in W. Stallings, *Practical Cryptography for Data Internetworks,* IEEE Computer Society Press, 1996, pp. 31–79.

A List of primitives

This appendix lists selected additive stream ciphers, hash functions, and block ciphers and their properties. For the additive stream ciphers and block ciphers, the best attack which is currently known is indicated. The primitives are listed in alphabetical order.

A.1 Additive stream ciphers

The following abbreviations are used:

k key length (in bits)
l length of the key-stream produced in one iteration (in bits)

Below the additive stream ciphers:

(alleged) RC4: $k = 8$ to 2048, $l = 8$. The memory consists of one table, containing the 256 possible values of a byte. A class of weak keys for RC4 has been identified [76]. Recently a statistical weakness in the key-stream has been exploited [32].

SEAL: $k = 160$, $l = 128$ [74, 75]. The memory consists of 2 tables with respectively 512 and 256 32-bit words, 1 table containing 16 bytes for every 1 Kbyte of output, and four registers. The best attack on SEAL uses 2^{32} samples of the key-stream to determine parts of the key dependent table [34].

WAKE: $k = 128$, $l = 32$ [90]. The memory consists of a table with 256 32-bit words and four registers. The original WAKE was later renamed WAKE-CFB, and can be broken with a differential attack. WAKE-OFB has been proposed; no attacks are known. WAKE-OFB has been extended to Wider-WAKE in [18].

A.2 Hash functions

The following abbreviations are used:

n length of hash result (in bits)
m length of message blocks (in bits)
R number of rounds
S number of steps

Below the list of hash functions. For a more extensive list and a reference to the best known attacks, the reader is referred to [65].

MD2: $n = 128$, $m = 128$, $R = 18$ [37].
MD4: $n = 128$, $m = 512$, $S = 48$ [70]. Extended MD4: $n = 256$ and $S = 2 \times 48$.

MD5: $n = 128$, $m = 512$, $S = 64$ [71].
RIPEMD-128: $n = 128$, $m = 512$, $S = 2 \times 64$ [12, 27].
RIPEMD-160: $n = 160$, $m = 512$, $S = 2 \times 80$ [12, 27].
SHA-1: $n = 160$, $m = 512$, $S = 80$ (+64 operations on data words) [30].
Snefru: $n = 128$ or 256, $m = 512 - n$, $R \geq 8$ [57].
Subhash: $n = 128$, $m = 32$, $R = 8$ [22].
Tiger: $n = 192$, $m = 512$, $R \geq 24$ [5].

A.3 Block ciphers

The following abbreviations are used:

l block length (bits)
k key length (bits)
R number of rounds
F or U the type of the block cipher: F stands for Feistel network, U stands for uniform transformation

For theoretical attacks the work for the required plaintext-ciphertext pairs is not counted.

Blowfish: $l = 64$, $k \leq 448$, $R = 16$, F [80].
Blowfish uses key dependent S-boxes. If the boxes are known, there is a differential attack [89] that works for a fraction of 2^{-17} of the keys. It requires 2^{51} chosen plaintexts, a memory of 2^{32}, and an effort of about 2^{57} encryptions. The attack breaks eight rounds for any key with 2^{48} chosen plaintexts, a memory of size 2^{32}, and an effort of about 2^{45} encryptions. A key recovery attack on four rounds is described in [66].

CAST: $l = 64$, $k = 64$ to 128, $R = 8$ to 16, F [1–4].
CAST is actually a design procedure. Ciphers designed according to this procedure inherit its name. An attack on eight rounds is described in [69] The attack requires 2^{62} chosen plaintexts. For CAST [1, 4] with 16-bit round keys the attack requires a memory of size 2^{16}, and an effort of about 2^{75} encryptions. For CAST with 32-bit round keys or 37-bit round keys [2] this becomes a memory of size 2^{32} or size 2^{37}, and an effort of about 2^{91} or 2^{96} encryptions respectively. For versions reduced to six rounds, the attack becomes very practical.

DES: $l = 64$, $k = 56$, $R = 16$, F [29].
The best theoretical attack is the linear attack [52], requiring 2^{43} known plaintexts, a memory of size 2^{13} and an effort of about 2^{19} encryptions. Because of the short key, exhaustive key search is feasible.

FEAL: $l = 64$, $k = 64$, $R = 8$, 16, 24, or 32, F [84].
The best attack on eight rounds is a differential-linear attack [7]. It requires 12 (twelve) chosen plaintexts, 2^{18} bytes of memory and has a workload of 35 days computer time. The versions with 16, 24, or 32 rounds are vulnerable to a differential attack that requires $2^{30}, 2^{46}$, or 2^{67} chosen plaintexts and has a workload of $2^{30}, 2^{46}$, or 2^{67} encryptions [8]. The 16-round and 24-round versions are also vulnerable to a linear attack, requiring 2^{19} or 2^{42} known plaintexts [60]. The differential attacks also apply to FEAL-NX [59].

GOST: $l = 64$, $k = 256$, $R = 32$, F [16]. The S-boxes of GOST are not specified. The only published attack is a related key chosen plaintext attack [38]. The probability of the differential characteristic depends on the S-boxes. Over a large set of randomly selected S-boxes this probability varies between 2^{-43} and 2^{-80} for a characteristic that may be used in an attack on twelve rounds.

IDEA: $l = 64$, $k = 128$, $R = 8(8.5)$, F (generalized). The output transformation is sometimes counted as an extra half-round [46].

The best attack is a truncated differential attack on three rounds including the output transformation [10]. 1% of the keys can be recovered using 2^{40} chosen plaintexts, a memory of 2^{48} and an effort of about 2^{51} encryptions. To find 31% of the keys, 2^{48} chosen plaintexts are required, the same amount of memory and an effort of 2^{59} encryptions.

Khafre: $l = 64$, $k = 64$ or 128, $R = 16, 24, 32, \ldots$, F [56].

The best attack is a chosen plaintext attack on 24 rounds [8]. It requires 2^{53} chosen plaintexts, a memory of 2^8, and one hour computer time.

Khufu: $l = 64$, $k \leq 512$, $R = 16, 24, 32, \ldots$, F [56].

The best attack is a chosen plaintext attack on 16 rounds [31]. It requires 2^{43} chosen plaintexts, a memory of 2^{25}, and an effort of about 2^{43} operations.

LOKI91: $l = 64$, $k = 64$, $R = 16$, F [15].

The best chosen plaintext attack on LOKI91 breaks 13 rounds [40]. It requires 2^{58} chosen plaintexts. The memory requirements are negligible and the effort is about 2^{48} encryptions. The best known plaintext attack breaks 12 rounds [79]. It requires 2^{63} known plaintexts, a memory of 2^{21}, and an effort of about 2^{63} encryptions.

MISTY: $l = 64$, $k = 128$, $R = 8, 12$, F (generalized) [53]. There are actually two MISTY algorithms: MISTY1 is an eight-round Feistel cipher with a new kind of key addition, MISTY2 has a more generalized Feistel structure and has 12 rounds.

To the best of our knowledge, no attacks have been published.

RC5: Everything is variable. The "nominal" values for the parameters are $l = 64$, $k = 128$, $R = 12$, U [72]. Every round of RC5 is composed of two 'almost' Feistel rounds.

The best attack on this version is a chosen plaintext attack [44]. It requires 2^{54} chosen plaintexts and a small amount of memory and work. For some weak keys, these numbers are reduced. If l is increased to 128, an attack on 24 rounds would need 2^{123} chosen plaintexts.

REDOC-II: $l = 70$, $k = 70$, $R = 10$, U [21].

The best attack is a chosen plaintext attack on four rounds [8]. It requires 2^{66} chosen plaintexts and a memory of size 2^{15}.

SAFER: $l = 64$, $k = 64$ or 128, $R = 6$, 8, or 10, U [50, 51]. Currently there are four versions of SAFER published: SAFER K-64 has $k = 64$ and $R = 6$, SAFER SK-64 has $k = 64$ and $R = 8$, SAFER K-128 and SAFER SK-128 have $k = 128$ and $R = 10$.

The key scheduling of the K versions has a weakness [42]. The best attack is a truncated differential attack on five rounds of SAFER K-64 [43]. It requires 2^{45} chosen plaintexts, a memory of 2^{32} and has a workload of 2^{46}

encryptions. The attack also recovers 32 key bits of SAFER K-128, reduced to five rounds.

SHARK: $l = 64$, $k = 64$ to 128, $R \geq 6$, U [67].
The best attack is a structure attack on three rounds. It requires 2^9 chosen plaintexts, a memory of size 2^8 and has a workload of 2^{17} encryptions.

SQUARE: $l = 128$, $k = 128$, $R \geq 8$, U [25, 26].
The best attack is a structure attack on six rounds [25]. It requires 2^{32} chosen plaintexts, a memory of size 2^{32} and has a workload of 2^{73} encryptions.

3DES: $l = 64$, $k = 112$ or 168, $R = 48$, F. 3DES consists of three DES encryptions in cascade. The three DES operations can use three different keys, or the first and the last key can be equal [87].
The best attack on three-key 3DES is the meet-in-the-middle attack: it uses two known plaintexts, a memory of 2^{56}, and has a workload of 2^{112} encryptions. The best chosen plaintext attack on two-key 3DES requires 2^{56} chosen plaintexts, a memory of size 2^{56}, and has a workload of 2^{56} encryptions [55]. The best known plaintext attack involves a trade-off [88]. When given 2^n known plaintexts, it requires a memory of 2^{56} and a work effort of about 2^{120-n} encryptions. See [41] for more details on multiple encryption.

3-WAY: $l = 96$, $k = 96$, $R = 11$, U [24].
To the best of our knowledge, no attacks have been published.