

RSA LABORATORIES' CryptoBytes

The technical newsletter of RSA Laboratories, a division of RSA Data Security, Inc.

Contents

1

Performance
Comparison of Public-
Key Cryptosystems

2

Editor's Note

6

Smart Card
Crypto-Coprocessors
for Public-Key
Cryptography

11

Standards Update

12

Chaffing and
Winnowing:
Confidentiality without
Encryption

18

DES, Triple-DES
and AES

24

Announcements

Performance Comparison of Public-Key Cryptosystems

Michael J. Wiener

Entrust Technologies
750 Heron Road
Ottawa, Ontario
Canada K1V 1A7

It has become clear over the past several years that public-key cryptography is an indispensable tool for simplifying key management and enabling secure communication. What is less clear is which of the available public-key cryptosystems is best. This article addresses this question taking into account such factors as security, computation speed, key size, and the intended application.

Main contenders

The most important consideration in choosing among public-key technologies is security. In some sense this is a choice among "religions". Do you believe most in the difficulty of factoring, computing discrete logarithms modulo a prime, or computing elliptic curve logarithms? Elliptic curve logarithms are the least studied at this point, but interest is rising.

There are three main public-key cryptosystem contenders. Each has a variable key size that can be increased to achieve higher security at the cost of slower cryptographic operations. The best attack known on each public-key cryptosystem requires an amount of computation determined by a security parameter which is related to the key size.

Michael Wiener is a senior cryptologist at Entrust Technologies. He can be contacted via e-mail at wiener@entrust.com.

RSA

The Rivest-Shamir-Adleman (RSA) public-key cryptosystem [8] involves exponentiation modulo a number n that is the product of two large prime numbers. When referring to the key size for RSA, what is meant is the length of the modulus n in bits. A typical key size for RSA is 1024 bits. The best attack known on RSA is called the General Number Field Sieve (GNFS) [4] which factors the modulus into the original prime numbers.

DH/DSA

Diffie-Hellman (DH) key exchange [1] and the Digital Signature Algorithm (DSA) [2] are public-key techniques based on exponentiation modulo a large prime number p . For these schemes, the key size is the length of the prime p in bits, and a typical value is 1024 bits. However, another important security parameter is the size of exponents used for exponentiation (assuming that this exponent is not larger than the order of the generator used). For DSA, the exponent size is fixed at 160 bits. For DH, exponents are often the same size as the prime p , but are frequently reduced from 1024 bits to between 160 and 256 bits, which is safe if appropriate precautions are taken [9]. When exploiting the size of the prime p , the best attack known is the General Number Field Sieve (adapted for discrete logarithms rather than factoring). When exploiting the size of exponents used, the best attack known is parallel collision search [10], which is based on Pollard's

(continued on page 3)



RSA Laboratories®

A Division of RSA Data Security

Editor's Note

In this, the Summer issue of *CryptoBytes*, we have four articles covering both well-known topics in cryptography and new ideas.

Michael Wiener leads off the issue with a comprehensive performance comparison of public-key cryptosystems. The article analyzes public-key cryptosystems with respect to factors such as security, computation speed, key size and the intended usage of the cryptosystem. The analysis makes it clear that there is no single "best" public-key technology, but that the best choice is situation dependent. Wiener's article is adapted from a talk he gave at the 1998 RSA Data Security Conference.

Helena Handschuh and Pascal Paillier follow this up with an article on smart card crypto-coprocessors for public-key cryptography. The article gives an extensive review of the technical characteristics and timings of the predominant smart card chips with crypto-coprocessors. It serves as a useful reference for the state of the art in crypto-coprocessors.

Ron Rivest presents a novel technique for ensuring the confidentiality of a message without encryption or steganography. The process is called chaffing and winnowing because the receiver must use message authentication codes in a winnowing process to remove the "chaff" that's been added to the message. This technique presents an especially timely alternative to encryption given the current controversy about law enforcement and cryptographic policy.

Finally, Eli Biham and Lars Knudsen provide an informative piece on DES, Triple-Des and NIST's search for an Advanced Encryption Standard (AES) to replace DES. The article provides an overview of the progress made in the cryptanalysis of DES and triple-DES, making the motivation behind the AES effort clear. The authors have submitted an AES candidate (together with Ross Anderson) called *Serpent*.

We also include an update on the PKCS standards and the IEEE P1363 standard project, "Standard Specifications for Public-Key Cryptography". Information on the upcoming PKCS workshop can be found here.

The future success of *CryptoBytes* depends on input from all sectors of the cryptographic community, and

as usual we would like very much to thank the writers who have contributed to this first issue of the first volume. We encourage any readers with comments, opposite opinions, suggestions or proposals for future issues to contact the *CryptoBytes* editor at RSA Laboratories or by E-mail to bytes-ed@rsa.com.

CryptoBytes Publication Schedule Changed

The *CryptoBytes* publication schedule has been revised. The newsletter will now be published as two, longer issues appearing in Summer and Winter of each year rather than the previous schedule of three, shorter issues. The newsletter will continue to be freely available from our website and copies of the newsletter will also be made available at major conferences.

Newsletter availability and contact information

CryptoBytes is a free publication and all issues, both current and past, are available via the World-Wide Web at [<http://www.rsa.com/rsalabs/pubs/cryptobytes.html>](http://www.rsa.com/rsalabs/pubs/cryptobytes.html).

For each previous issue a limited number of copies were printed. While available, copies of these printed issues can be requested by contacting RSA Laboratories though a nominal fee to cover handling costs may be charged for individual requests.

RSA Laboratories can be contacted at:

RSA Laboratories
2955 Campus Drive, Suite 400
San Mateo, CA 94403
650/295-7600
650/295-7599 (fax)
rsa-labs@rsa.com

About RSA Laboratories

An academic environment within a commercial organization, RSA Laboratories is the research and consulting division of RSA Data Security, the company founded by the inventors of the RSA public-key cryptosystem. Its purpose is to provide state-of-the-art expertise on cryptography and information security for the benefit of RSA Data Security and its customers. RSA Data Security is a Security Dynamics company.

**We encourage
any readers
with comments,
opposite opinions,
suggestions or
proposals for
future issues to
contact the
CryptoBytes
editor.**

Performance Comparison of Public-Key Cryptosystems

Continued from page 1

rho- and lambda-methods of computing discrete logarithms [7].

Elliptic curves

Elliptic curve cryptosystems are based on computations with points on an elliptic curve [3, 5]. These cryptosystems are variants of Diffie-Hellman and DSA (ECDH and ECDSA). Typically, elliptic curves are defined over either the integers modulo a prime number ($GF(p)$) or over binary polynomials ($GF(2^m)$). When referring to the key size, what is meant is the size of the prime number or binary polynomials in bits. Typical key sizes are in the range 160 to 200 bits. The security parameter is the size of multipliers which are analogous to the exponents in DH and DSA. The size of these multipliers is limited to the order of the generator used, which is slightly smaller than the key size. The best attack known on ECDH and ECDSA is to compute elliptic curve logarithms using parallel collision search [10].

Uses of public-key techniques

The three main uses of public-key techniques are digital signatures, encryption and decryption for passing symmetric keys, and on-line key exchange. All three public-key technologies can be used for each of these purposes.

Digital signatures

Digital signing of data can be performed using RSA signatures, DSA, or ECDSA.

Encryption/decryption

For store-and-forward communications such as email, encryption of the bulk user data is most efficiently done with a symmetric cryptosystem. The symmetric key is then encrypted with the public key of the intended recipient. This key management function can be done using RSA encryption, Diffie-Hellman (in "half-static" mode using one long-term key and one short-term key), or the elliptic curve variant of Diffie-Hellman.

On-line key exchange

For on-line communications such as web browsing, it is possible to encrypt the communications session with a key passed from one party to the other using a store-and-forward method described under *Encryp-*

tion/decryption. However, in the on-line case, it is possible to achieve a property called perfect forward secrecy, which means that if either party's long-term keys are compromised, then past sessions keys remain secure. Diffie-Hellman and ECDH provide this property when used in ephemeral mode where both keys are short-term. Typically, on-line key exchange is combined with digital signatures to give an authenticated key exchange. It is (incorrectly) widely believed that RSA cannot be used to give perfect forward secrecy. This only applies to the use of RSA in a store-and-forward mode. If a one-time RSA key pair is generated by one party who sends the public key to the other party, and the other party returns a symmetric key encrypted with the public key, then perfect forward secrecy is achieved, assuming that the RSA key pair is erased after its use.

Key sizes for comparable security levels

To fairly compare the speeds of the different public-key techniques, it is necessary to decide what key sizes give comparable security levels. These comparisons are necessarily based on the best attacks known. Any future improvements in attacks will change the results of the analysis. RSA with a 1024-bit modulus is used as the basis for comparison, and we determine what Diffie-Hellman, DSA, and elliptic curve key sizes give the same level of security.

Comparing RSA to Diffie-Hellman and DSA is fairly easy. The General Number Field Sieve (GNFS) requires about the same run-time to factor a 1024-bit RSA modulus and to perform a discrete logarithm with a 1024-bit Diffie-Hellman or DSA modulus.

Comparing RSA to elliptic curves is more difficult and requires a more detailed analysis. Odlyzko estimated that factoring a 1024-bit RSA modulus with GNFS requires 3×10^{11} MIPS-years [6] (a MIPS-year is one year of time on a computer performing one million instructions per second). Using an estimate of 300 MIPS for the latest generation of PCs gives an estimate to factor a 1024-bit RSA modulus of 2^{30} PC-years ($2^{30}/t$ years of elapsed time on t PCs). It may be possible to do better than this using custom hardware, but PCs are quite well-suited to factoring due to the high memory requirements. Combining this with the difficulty of keeping up with the level

The three main uses of public-key techniques are digital signatures, encryption and decryption for passing symmetric keys, and on-line key exchange. All three public-key technologies can be used for each of these purposes.

We can now compare the computation speeds of the different public-key techniques [...]. The results of this comparison differ depending upon which use of public-key cryptography is considered

The differences show up when considering RSA, particularly the very fast RSA signature verification

of technology in mass-produced PCs makes it doubtful that the use of custom hardware could improve significantly on PCs for factoring.

A parallel collision search attack on elliptic curves with k -bit multipliers (exponents) requires about $2^{k/2}$ elliptic curve additions. For a software-based attack with PCs, each elliptic curve addition requires about 2^{-14} second (2^{-39} year), giving a total attack time of $2^{k/2-39}$ PC-years. This is the same as the 2^{30} PC-years for factoring a 1024-bit RSA modulus when $k = 138$, a fairly small elliptic curve. However, attackers are not limited to attacking elliptic curves in software. Parallel collision search has very low memory requirements, and there is much to be gained by going to a hardware-based attack, even using technology which is less advanced than that used in PCs. A fully pipelined chip for elliptic curve additions over $GF(2^m)$ could be run at 64 MHz (2^{51} additions per year) and would cost about \$16. The total attack time is $2^{k/2-51}$ chip-years. We now have to compare the cost of a PC to the cost of a chip. It would not be fair to assume that the attacker would have to assume the full cost of a PC. On the other hand, it is not fair to assume that the attacker could access all PCs for free; people will donate computer time to what they deem to be worthy causes, but it is doubtful

that many people would donate computer time to break a single individual's key. We assume here that full-time access to a PC could be had for \$250, the cost of about 16 chips. To achieve the level of security of 1024-bit RSA with elliptic curves requires $k = 170$. To use multipliers of 170 bits requires an elliptic curve key size in the range 171-180 bits. The required key size for elliptic curves over $GF(p)$ is lower (by perhaps 8 or 10 bits) because elliptic curve addition for this case is more expensive to implement in hardware than it is for the $GF(2^m)$ case.

Computation speed comparison

We can now compare the computation speeds of the different public-key techniques for key sizes which give comparable security levels. The results of this comparison differ depending upon which use of public-key cryptography is considered: digital sig-

natures, encryption/decryption, and on-line key exchange.

Table 1 summarizes the computation times for the operations associated with digital signatures. For the most frequent operations, signing and signature verification, DSA and ECDSA over $GF(p)$ are comparable. However, RSA is slower for signing and much faster for signature verification. Elliptic curves over $GF(2^m)$ appear to be slower than those over $GF(p)$, but reports differ on this. There is a narrow class of curves over $GF(2^m)$, called anomalous binary curves or Koblitz curves which give faster computations, but recently these have been shown to be slightly less secure than first thought.

For public-key encryption and decryption operations, the comparison is similar to the digital signature comparison. Diffie-Hellman is comparable to ECDH over $GF(p)$, and RSA is slower for decryption and much faster for encryption. For on-line key ex-

	RSA-1024 (e = 3)	DSA-1024	ECDSA-168 (over GF(p))
sign	43	7	5
verify	0.6	27	19
key generation	1100	7	7
parameter generation	none	6500	large (research area)

Figure 1. Digital signature timings (milliseconds on a 200 MHz Pentium Pro)

change giving perfect forward secrecy, DH and ECDH are comparable, but RSA is much slower due to the need to create the one-time RSA key pair.

Although different benchmarks vary considerably, when the best available computation times are considered, elliptic curves are comparable to DSA/DH. The differences show up when considering RSA, particularly the very fast RSA signature verification, which is quite important in certificate-based systems where the most frequent public-key operation is verifying a signature.

Which public-key technology is best?

There is no single answer to the question of which public-key technology is best. The answer depends upon the setting in which public key methods are used.

Setting 1: certificate-based systems.

Certification Authority (CA) key pairs are used for

Smart Card Crypto-Coprocessors for Public-Key Cryptography

Helena Handschuh
Pascal Paillier

Gemplus
Cryptography Dept.
34, rue Guynemer
F-92447 Issy-Les-Mx.

ENST
Computer Science Dept.
46, rue Barrault
F-75634 Paris Cedex 13

In recent years, public key cryptography has gained increasing attention from both companies and end users who wish to use this technology to add security to a wide variety of applications. One consequence of this trend has been the growing importance of public key smart cards to store a user's private key and to provide a secure computing environment for the private key operation. As such, these smart cards are used as secure tokens.

Many chip manufacturers are therefore proposing ever better and faster implementations of public key algorithms using dedicated crypto-coprocessors on their chips. Two years ago the main focus of attention was on established public-key techniques such as RSA [10] and discrete-logarithm based cryptosystems [4, 6]. The main focus of attention was on the performance of these algorithms with key sizes that were lying around the 512-bit range. A survey of the state of the art at that time was published in [8]. This survey is intended to give an overview of recent evolutions in the crypto-coprocessor market and the latest achievements in terms of the speed and the range of modulus size that is available. We also include consideration of implementations of other public-key techniques such as elliptic curve cryptosystems [7].

The most widely used smart card chips with crypto-coprocessors are listed here. The usual manufacturers such as Thomson, Siemens, Philips and Motorola still lead the market, but new manufacturers have joined the public-key battlefield since 1996, such as NEC, Hitachi and Toshiba. Table 1 lists the standard field size of such chips in terms of on-board memory sizes (RAM, ROM, EEPROM), operating voltage and frequency, abnormal behaviour sensors (High or Low Voltage, Frequency or Temperature),

and the maximum public key size supported for RSA or DSA public modulus, and elliptic curve characteristic as appropriate.

It can be observed that new trends include increasing RAM and EEPROM sizes for user customized applications, faster internal clocks, improved security features such as new sensors or address scrambling matrices, and growing public moduli sizes. In other words, chips are becoming bigger, more versatile, faster and increasingly secure.

The new range of public key sizes for RSA and DSA is now generally up to 1024 bits, and some chips can even handle 2048-bit computations. But as the need for ever bigger moduli becomes less acute, one future direction for improvement is in a battle for increased performance. Public key algorithms are currently used only for authentication or access control as they are much too slow for stream encryption.

Every architecture has its own optimizations for computing modular multiplications and exponentiations which are the most important operations in most public key cryptosystems. See [8] for some examples. But basically most chips end up achieving comparable speed. Table 2 lists the computation times on different chips for different public key algorithms with the most widely implemented system being RSA. However, DSA [5] and the elliptic curve equivalent ECDSA, as well as other systems such as ESIGN appear in some implementations.

With regard to signing with RSA only 1024-bit RSA is currently possible with a performance time of less than a second, but we suspect that it will not be too long before 2048-bit RSA becomes practical as well. As a matter of fact, in one of our custom implementations, 2048-bit signatures can already be computed in a reasonable time (around 2 seconds) using the Chinese Remainder Theorem, and a signature verification using a small exponent (typically $e = F_4 = 2^{16} + 1$) which can be computed in about 360 ms applying what are called size-doubling methods (see below).

Modular arithmetic on CCPs

Modular multiplication, the computation of $Z = XY \bmod N$ is certainly the most useful operation in public key cryptography, and its efficiency in terms of hardware throughput remains highly critical for

[...] as the need for ever bigger moduli becomes less acute, one future direction for improvement is in a battle for increased performance.

Both authors jointly work with Gemplus and ENST. Helena Handschuh can be contacted by e-mail at handschuh@gemplus.com or handschu@enst.fr. Pascal Paillier can be contacted by e-mail at paillier@gemplus.com or paillier@enst.fr.

Name	Manufacturer	µC-core Name	CCP Modulus	Max	RAM	ROM	EEPROM	Voltage	Max Ext. Clock	Max Int. Clock	Tech-nology	Sensors
H8/3111	Hitachi	H8/300	Coprocessor	576 bit	800 B	14 KB	8 KB	3V & 5V	10 Mhz	10 Mhz	0.8 µm	LV, LF
H8/3112	Hitachi	H8/300	Coprocessor	576 bit	1312 B	24 KB	8 KB	3V & 5V	10 Mhz	10 Mhz	0.8 µm	LV, LF, HF
H8/3113*	Hitachi	H8/300	Coprocessor	1024 bit	1.5 KB	32 KB	16 KB	3V & 5V	10 Mhz	14.32 Mhz	0.5 µm	LV, HV, LF, HF
T6N29	Toshiba	Z80	1024B	1024 bit	512 B	20 KB	8 KB	3V & 5V	—	—	0.6 µm	V
T6N37*	Toshiba	Z80	1024B	1024 bit	512 B	20 KB	8 KB	3V & 5V	—	—	—	V/T/F
T6N39*	Toshiba	Z80	1024B	1024 bit	512 B	20 KB	8 KB	3V & 5V	—	—	—	V/T/F
T6N42*	Toshiba	Z80	2048B	2048 bit	512 B	20 KB	8 KB	3V & 5V	—	—	—	V/T/F
ST16CF54B	SGS-Thomson	8 bit MCU	MAP	512 bit	512 B	16 KB	4 KB	5V +/- 10%	5 Mhz	5 Mhz	—	—
ST19CF68	SGS-Thomson	8 bit CPU	MAP	512 bit	960 B	23 KB	8 KB	3V,5V +/- 10%	10 Mhz	10 Mhz	0.6 µm	—
ST19KF16	SGS-Thomson	8 bit CPU	MAP	1088 bit	960 B	32 KB	16 KB	3V,5V +/- 10%	10 Mhz	10 Mhz	0.6 µm	—
P83W854	Philips	80C51	FameX	2048 bit	800 B	20 KB	4 KB	2.7V to 5.5V	8 Mhz	—	—	V/T/F
P83W858	Philips	80C51	FameX	2048 bit	800 B	20 KB	8 KB	2.7V to 5.5V	8 Mhz	—	—	V/T/F
P83W8516	Philips	80C51	FameX	2048 bit	2304 B	32 KB	16 KB	2.7V to 5.5V	8 Mhz	—	—	V/T/F
P83W8532	Philips	80C51	FameX	2048 bit	2304 B	32 KB	32 KB	2.7V to 5.5V	8 Mhz	—	—	V/T/F
SmartXA	Philips	16 bit CPU	FameX	2048 bit	1.5/2 KB	32 KB	8/16/32 KB					
SLE44CR80S	Siemens	80C51	CCP	540 bit	256 B	17 KB	8 KB	3V to 5V	7.5 Mhz	7.5 Mhz	0.7 µm	—
SLE66CX160S	Siemens	80C51	ACE	1100 bit	1280 B	32 KB	16 KB	2.7V to 5.5V	7.5 Mhz	7.5 Mhz	0.6 µm	—
µPD789828*	NEC	78K0S	SuperMAP	2048 bit	1 KB	24 KB	8 KB	1.8V to 5.5V	5 Mhz	40 Mhz	0.35 µm	—

* expected in forthcoming months

modular exponentiation-based schemes [34]. There exist many different ways of performing a modular multiplication, among which Montgomery [7], De Waleffe & Quisquater [13], or Sedlak [12] provide methods that are still the most commonly used in hardware implementations. Naturally, the best internal architecture of a coprocessor relies strongly on the choice of modular multiplication algorithm (see [6, 9] for details).

Size-doubling techniques : algorithmic tools for emulating a 2n-bit CCP from an n-bit one

There exist some tricky algorithmic techniques that allow one to virtually increase (typically double) the size of the crypto-coprocessor being used. Basically, these techniques allow the execution of a 1024-bit (resp. 2048-bit) exponentiation on a 512-bit (resp. 1024-bit) processor. These techniques use an additional API layer which implements an up-to-2n-bit arithmetic engine out of an n-bit one (*this technique is covered by a Gemplus patent*), thereby providing up-to-2n-bit modular addition, multiplication, and exponentiation. Applied to a 1024-bit processor, this would allow 2048-bit DSA, 2048-bit Diffie-Hellman Key Exchange, 2048-bit ESIGN, 2048-bit RSA verification, 4096-bit RSA signature generation, and so on.

A particularly interesting case is the implementation of a fast 2048-bit version of RSA on a 1024-bit chip. As only a few manufacturers have planned 2048-bit platforms so far, this seems to be the most useful context in which to apply such size-doubling techniques. These arithmetic algorithms rely on several simultaneous computation strategies including:

1. modular representation of long operands,
2. low-storage Chinese Remainder Theorem combinations,
3. new parallelizable Montgomery-like operations, and
4. fast modular computation techniques.

The techniques make previously unavailable operations on large numbers (2048-bit regular multiplication, 2048-bit addition, 2048-bit modular addition, multiplication and squaring) available and at the disposal of public key cryptographic algorithms so that they can be used with a larger key size.

Modular representations

Long operands (ranging from 1025 to 4096 bits) are usually given under a radix form in base 2^{1024} . To be computationally exploitable, they are at first splitted into (at most) four workable 1024-bit words

Table 1: Technical characteristics of the CCPs

Table 2 : Public
key algorithm
timings

Chip		H8/3111-3112	H8/3113	ST16CF54B	ST19CF68	ST19KF16
Internal Clock Frequency		3.57 Mhz	14 Mhz	5 Mhz	10 Mhz	10 Mhz
DES	64 bits	N/A	N/A	10 ms*	N/A	N/A
SHA	512 bits	N/A	N/A	15.2 ms	8.2 ms	8.2 ms
MD5	512 bits	N/A	N/A	12 ms*	N/A	N/A
RSA 512	Sign with CRT	202 ms	N/A	142 ms	70 ms	20 ms
RSA 512	Sign without CRT	514 ms	68 ms	389 ms	195 ms	55 ms
RSA 512	Verify (e = F ₄)	N/A	N/A	9 ms	4.5 ms	2 ms
RSA 768	Sign with CRT	N/A	N/A	377 ms	189 ms	50 ms
RSA 768	Sign without CRT	N/A	210 ms	N/A	N/A	165 ms
RSA 768	Verify (e = F ₄)	N/A	N/A	190 ms	100 ms	3 ms
RSA 1024	Sign with CRT	N/A	N/A	800 ms	400 ms	110 ms
RSA 1024	Sign without CRT	N/A	480 ms	N/A	N/A	380 ms
RSA 1024	Verify (e = F ₄)	N/A	N/A	265 ms	150 ms	5 ms
RSA 2048	Sign with CRT	N/A	N/A	N/A	N/A	780 ms
RSA 2048	Sign without CRT	N/A	N/A	N/A	N/A	N/A
RSA 2048	Verify (e = F ₄)	N/A	N/A	N/A	N/A	100 ms
DSA 512**	Sign	N/A	N/A	163 ms	84 ms	25 ms
DSA 512**	Verify	N/A	N/A	283 ms	146 ms	40 ms
DSA 768**	Sign	N/A	N/A	N/A	N/A	50 ms
DSA 768**	Verify	N/A	N/A	N/A	N/A	80 ms
DSA 1024**	Sign	N/A	N/A	N/A	N/A	100 ms
DSA 1024**	Verify	N/A	N/A	N/A	N/A	160 ms
ECDSA 135/131	Sign	N/A	N/A	N/A	N/A	N/A
ECDSA 135/131	Verify	N/A	N/A	N/A	N/A	N/A
ECDSA 255	Sign	N/A	N/A	N/A	N/A	N/A
ECDSA 255	Verify	N/A	N/A	N/A	N/A	N/A

* according to Gemplus' implementation ** in all the DSA entries, the subgroup size is 160 bits

$$X \rightarrow \langle X \rangle = (X \bmod a_1, X \bmod a_2, \\ X \bmod a_3, X \bmod a_4),$$

where a_1, a_2, a_3 and a_4 are relatively prime easy-to-generate 1024-bit constants. Clearly the representation conversion $X \rightarrow \langle X \rangle$ consists solely of modular reductions with a 1024-bit modulus, which is already implemented and directly available from the underlying arithmetic engine. Larger data appears to be much easier to handle under such a form, as computing

$$\langle X \rangle, \langle Y \rangle \rightarrow \langle X + Y \rangle, \quad (1)$$

or

$$\langle X \rangle, \langle Y \rangle \rightarrow \langle XY \rangle, \quad (2)$$

is carry-free and can be performed by independent parallel computations. It is worthwhile noticing that

multiplication (2) presents a lower complexity than would a 2n-bit classical multiplication since :

$$\Sigma (\log a_i) < (\log \Pi a_i).$$

Hence, this representation leads to time savings when compared with the classical radix-form oriented manipulations. Obviously one has to be careful that the cost of recombining the operands afterwards does not reduce the newly gained benefit too much. A set of moduli $\{a_1, a_2, a_3, a_4\}$ that offers this property is said to be *low-cost*.

Fast CRT combinations

The original form of operands is re-obtained, after various calculations, by combining coordinates according to Garner's CRT algorithm

$$X \bmod a_1 a_2 = ((X \bmod a_2 - X \bmod a_1) a_1^{-1} \bmod a_2) a_1 + X \bmod a_1$$

which can be cascaded to obtain a modular-to-radix transformation

$$\langle X \rangle = (X \bmod a_1, X \bmod a_2, X \bmod a_3, X \bmod a_4) \rightarrow X.$$

The use of low-cost moduli allows the recombination of the correct operands at almost no extra cost (only linear operations). This allows for high-speed representation conversion.

Modular Montgomery reduction

Once two operands $\langle X \rangle$ and $\langle Y \rangle$ are given under their modular representation, one will typically wish

to compute $\langle XY \bmod N \rangle$ where N may be up-to-2048-bit long. For that purpose, a newly discovered efficient algorithm for computing

$$\langle X \rangle, \langle Y \rangle, \langle N \rangle \rightarrow \langle XY \bmod N \rangle$$

Table 3: Key size
in size-doubling
techniques

mains fully compatible with exponent-relative precomputations (signed-digit, redundant representations, $d+k\phi(N)$, and so forth).


As far as we know, only this size-doubling technique makes it possible to execute on board a 1024-bit smart card any personalized version of RSA with keysize ranging continuously from 1025 to 2048 bits, and while using any public exponent. The required key material (N , e , constants) is proportional in length to the chosen RSA-size and Table 3 gives an illustration of the memory requirements for implementing an n -bit version of RSA ($1025 \leq n \leq 2048$).

E ² PROM (Key size)	0.875 x n bytes
Required RAM	0.875 x n bytes

Conclusion

In this brief article we have presented the data obtained during a study of the current state-of-the-art public-key cryptosystem implementations on smart cards. We have also added some information on the performance of the secret-key algorithm DES and some popular hash functions such as SHA-1 and MD5. Although most of the performance improvements are still made to the public key implementations, some chips already feature dedicated secret-key coprocessors. One question for the future is whether it will be possible to embed both on the same chip, and/or whether 16-bit and 32-bit processors (like the CASCADE chip [2]) will spark a revolution in the developing smart card market.

Acknowledgements

We would like to give special thanks to all interested parties who kindly helped us to gather the technical details for our survey, particularly those that could not be found in the usual documentation. We have listed some contact names and phone numbers that the reader might find useful. 

References

- [1] P. Barrett, *Implementing the Rivest, Shamir and Adleman public-key encryption algorithm on a standard digital signal processor*, Advances in Cryptology: CRYPTO'86 (A. M. Odlyzko ed.), LNCS 263, Springer-Verlag, pp. 311-323, 1987.
- [2] J. F. Dhem, *Design of an efficient public-key cryptographic library for RISC-based smart cards*, PhD Thesis, UCL, 1998.
- [3] W. Diffie and M. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, vol IT-22, no. 6, pp. 644-654, November 1976.
- [4] T. El-Gamal, *A public-key cryptosystem and a signature scheme based on discrete logarithms*, IEEE TIT, vol. IT-31:4, pp 469-472, 1985.
- [5] A. Fiat and A. Shamir, *How to prove yourself: Practical solutions to identification and signature problems*, Advances in Cryptology: CRYPTO'86 (A. M. Odlyzko ed.), LNCS 263, Springer-Verlag, pp. 181-187, 1987.
- [6] D. Knuth, *The Art of Computer Programming*, vol. 2, Seminumerical Algorithms, pp. 248-250, 1969.
- [7] V. S. Miller, *Use of Elliptic Curves in Cryptography*, Advances in Cryptology: CRYPTO'85 proceedings, Springer-Verlag 1986, pp. 417-426.
- [8] P. Montgomery, *Modular multiplication without trial division*

(continued at bottom of next page)

Company	Contact name	Phone/fax no.	Address	E-mail address
Hitachi	Nicolas Prawitz	+ 33 1 34630500 + 33 1 34653431	18, rue Grange Dame Rose, BP 134 F-78148 Velizy Cedex	Nicolas.Prawitz@hitachi-eu.com
Motorola	Dan Muenchau	+ 1 972 5165134		
NEC	Nicolas Bérard	+ 33 1 30675800 + 33 1 30675899	9, rue Paul Dautier F-78142 Velizy Villacoublay	BerardN@ef.nec.de
Philips GmbH	Thomas Wille	+ 49 40 56133690 + 49 40 56133045	Röhren- und Halbleiterwerke PO Box 540240 D-22502 Hamburg	
SGS-Thomson	Frédéric Barbato	+ 33 1 47407575 + 33 1 47407910	7, avenue Gallieni, BP 93 F-94253 Gentilly Cedex	Frederic.Barbato@st.com
Siemens	Laurent Deloo	+ 33 1 49223100 + 33 1 49223413	39-47, boulevard Ornano F-93527 Saint-Denis Cedex 2	Laurent.Deloo@p1.siemens.fr

PKCS Standards

PKCS #1

PKCS #1 is currently being extensively revised. A new draft and a bulletin giving more information (prepared jointly with Daniel Bleichenbacher of Bell Laboratories) can be found at <http://www.rsa.com/rsalabs>.

PKCS #11

Version 2.01 of the PKCS #11: Cryptographic Token Interface Standard (Cryptoki) standard, defining an application-level interface for programs to use cryptographic devices, was released in December 1997.

PKCS #5 and PKCS #14

The development of PKCS #5 (Password-Based Encryption) v2.0 and PKCS #14 (Pseudorandom Number Generation) is proceeding in parallel, and drafts are planned for Q3. The current effort focuses on establishing provable, yet efficient ways of transforming a seed (be it a password, inter-keystroke timings, or some other non-uniformly-distributed random source) into a uniformly-distributed key. This key then feeds into an encryption or authentication scheme (in the case of PKCS #5) or a Pseudorandom Number Generator (in the case of PKCS #14). Both standards will include support for variable-size seed material, variable-size keys, and a variety of cryptographic primitives.

PKCS Workshop Announcement

The next PKCS standards workshop will be held this fall. The topics to be covered are:

- PKCS #1 (RSA) and #13 (ECC)
- PKCS #5 (Password-based encryption) and #14 (PRNG)
- PKCS #15 (Smart Card file format)
- General planning and updates on PKCS #7 v2, PKCS #11, PKCS #12

The exact dates and the location of the workshop will be posted at <http://www.rsa.com/rsalabs/pubs/PKCS> when it is available.

IEEE P1363

The IEEE P1363 project, "Standard Specifications for Public-Key Cryptography," continues to move toward completion, with balloting expected later this year. The project, now in its fifth year, has produced a comprehensive draft standard covering public-key techniques from the discrete logarithm, elliptic curve, and integer factorization families. Contributions are currently solicited for an addendum, IEEE P1363a, which will cover additional public-key techniques.

The project is closely coordinated with emerging ANSI standards for public-key cryptography in banking, and forthcoming revisions of RSA Laboratories' Public-Key Cryptography Standards will also be aligned with IEEE P1363. For more information, see <http://grouper.ieee.org/groups/1363/>.

Smart Card Crypto-Coprocessors for Public-Key Cryptography

Continued from page 10

- sion, *Mathematics of Computation*, vol. 44(170), pp. 519-521, 1985.
- [9] D. Naccache and D. M'Raihi, *Arithmetic co-processors for public-key cryptography: The State of the Art*, IEEE Micro, pp. 14-24, June 1996.
- [10] NIST. FIPS PUB 186, February 1, 1993, *Digital Signature Standard*.
- [11] J. Omura, *A Public Key Cell Design for Smart Card Chips*, IT Workshop, Hawaii, USA, November 27-30, 1990, pp. 983-985.
- [12] R. Rivest, A. Shamir and L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. Communications of the ACM, vol. 21, no. 2, p.120-126, February 1978.
- [13] C. Schnorr, *Efficient identification and signatures for smart-cards*, *Advances in Cryptology: EUROCRYPT'89* (G. Brassard ed.), LNCS 435, Springer-Verlag, pp. 239-252, 1990.
- [14] H. Sedlak, *The RSA cryptographic Processor: The First High Speed One-Chip Solution*, *Advances in Cryptology: EUROCRYPT'87* (C. Pomerance ed.), LNCS 293, Springer-Verlag, pp. 95-105, 1988.
- [15] D. de Waleffe and J.J. Quisquater, *CORSAIR, a smart card for public-key cryptosystems*, *Advances in Cryptology: CRYPTO'90* (A. Menezes and S. Vanstone ed.), LNCS 537, Springer-Verlag, pp. 503-513, 1990.

Chaffing and Winnowing: Confidentiality without Encryption

Ronald L. Rivest

MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

A major goal of security techniques is “confidentiality”—ensuring that adversaries gain no intelligence from a transmitted message. There are two major techniques for achieving confidentiality:

- **Steganography:** the art of hiding a secret message within a larger one in such a way that the adversary can not discern the presence or contents of the hidden message. For example, a message might be hidden within a picture by changing the low-order pixel bits to be the message bits. (See Wayner (1996) for more information on steganography.)
- **Encryption:** transforming the message to a ciphertext such that an adversary who overhears the ciphertext can not determine the message sent. The legitimate receiver possesses a secret decryption key that allows him to reverse the encryption transformation and retrieve the message. The sender may have used the same key to encrypt the message (with symmetric encryption schemes) or used a different, but related key (with public-key schemes). DES and RSA are familiar examples of encryption schemes.

This paper introduces a new technique, which we call “chaffing and winnowing”—to winnow is to “separate out or eliminate (the poor or useless parts),” (Webster’s Dictionary), and is often used when referring to the process of separating grain from chaff.

Novel techniques for confidentiality are interesting in part because of the current debate about cryptographic policy as to whether law enforcement should be given when authorized surreptitious access to the plaintext of encrypted messages. The usual technique proposed for such access is “key recovery,” where law enforcement has a “back door” that enables them to recover the decryption key.

Professor Ronald Rivest is associate director of MIT’s Laboratory for Computer Science. He can be contacted at rivest@theory.lcs.mit.edu. The text of this article can also be found at <http://theory.lcs.mit.edu/~rivest/chaffing.txt>

Winnowing does not employ encryption, and so does not have a “decryption key.” Thus, the usual arguments in favor of “key recovery” don’t apply very well for winnowing. As usual, the policy debate about regulating technology ends up being obsoleted by technological innovations. Trying to regulate confidentiality by regulating encryption closes one door and leaves two open (steganography and winnowing).

We now explain how a confidentiality system based on winnowing works. There are two parts to sending a message: authenticating (adding MACs), and adding chaff. The recipient removes the chaff to obtain the original message.

The sender breaks the message into packets, and authenticates each packet using a secret authentication key. That is, the sender appends to each packet a “message authentication code” or “MAC” computed as a function of the packet contents and the secret authentication key, using some standard MAC algorithm, such as HMAC-SHA1 (see Krawczyk et al. (1997)). We have the transformation of appending a MAC thus:

packet → packet, MAC

The packet is still “in the clear”; no encryption has been performed. We note that software that merely authenticates messages by adding MACs is automatically approved for export, as it is deemed not to encrypt.

There is a secret key shared by the sender and the receiver to authenticate the origin and contents of each packet—the legitimate receiver, knowing the secret authentication key, can determine that a packet is authentic by recomputing the MAC and comparing it to the received MAC. If the comparison fails, the packet and its MAC are automatically discarded. The sender and the receiver can initially create and agree upon the secret authentication key with any standard technique, such as authenticated Diffie-Hellman.

We note that it is typical for each packet to contain a serial number as well. For example, when a long file is transmitted it is broken up into smaller packets, and each packet carries a unique serial number. The serial numbers help the receiver to remove duplicate

Novel techniques for confidentiality are interesting in part because of the current debate about cryptographic policy

Trying to regulate confidentiality by regulating encryption closes one door and leaves two open (steganography and winnowing).

packets, identify missing packets, and to correctly order the received packets when reassembling the file. The MAC for a packet is computed as a function of the serial number of the packet as well as of the packet contents and the secret authentication key. As an example, we might have a sequence of the form:

```
(1,Hi Bob,465231)
(2,Meet me at,782290)
(3,7PM,344287)
(4,Love-Alice,312265)
```

of triples of sequence number, message, and MAC.

The second process involved in sending a message is “adding chaff”: adding fake packets with bogus MACs. The chaff packets have the correct overall format, have reasonable serial numbers and reasonable message contents, but have MACs that are not valid. The chaff packets may be randomly intermingled with the good (wheat) packets to form the transmitted packet sequence. Extending the preceding example, chaff packets might make the received sequence look like:

```
(1,Hi Larry,532105)
(1,Hi Bob,465231)
(2,Meet me at,782290)
(2,I'll call you at,793122)
(3,6PM,891231)
(3,7PM,344287)
(4,Yours-Susan,553419)
(4,Love-Alice,312265)
```

In this case, for each serial number, one packet is good (wheat) and one is bad (chaff). Instead of randomly intermingling the chaff with the wheat, the packets can also be output in sorted order, sorting first by serial number, and then by message contents.

To obtain the correct message, the receiver merely discards all of the chaff packets, and retains the wheat packets. But this is what the receiver does anyway! In a typical packet-based communication system the receiver will automatically discard all packets with bad MACs. So the “winnowing” process is a normal part of such a system. (Receiving a packet with a bad MAC could conceivably trigger more of a response from the receiver, but not normally; the detection of a missing packet is deter-

mined at a different level of the protocol stack, rather than upon receipt of a bad packet, since the packet may have been transmitted more than once and been received OK already.)

Let us verb a word, and let “chaffing” mean the process of adding chaff to a sequence of packets. As above, “winnowing” is the (usual) process of discarding all packets with bad MACs. We call the good packets “wheat” for consistency of metaphor.

How much confidentiality does chaffing provide? This depends on the MAC algorithm, on how the original message is broken into packets, and on how the chaffing is done.

A typical MAC algorithm (such as HMAC-SHA1) will appear to act like a “random function” to the adversary, and in such a case the adversary will not be able to distinguish wheat from chaff. It is possible in principle, however, to have an unfortunate MAC algorithm that “leaks” information about the message being MAC’ed, allowing the adversary to gain an advantage in distinguishing wheat from chaff. For example, one could define a LEAKY-HMAC-SHA1 MAC algorithm to have an output that is the concatenation of the output of the HMAC-SHA1 algorithm together with the low-order bit of the message being MAC’ed. However, in practice (and in theory) one looks for MAC algorithms that are indistinguishable from random functions, and such algorithms also work fine in a chaffing and winnowing application.

Note that the problem of providing confidentiality by chaffing and winnowing is based on the difficulty (for the adversary) of distinguishing the chaff from the wheat. It is *not* based on the difficulty of breaking an encryption scheme, since there is no encryption being performed (although confidentiality may be obtained nonetheless, just as for steganography).

If the adversary sees only one packet with a given serial number, then that packet is probably wheat, and not chaff. So a good chaffing process will add at least one chaff packet for each packet serial number used by the message.

The adversary may also distinguish wheat from chaff by the contents of each packet. If the wheat packets

Note that the problem of providing confidentiality by chaffing and winnowing is based on the difficulty (for the adversary) of distinguishing the chaff from the wheat. It is not based on the difficulty of breaking an encryption scheme

I stress that the sending process for chaffing and winnowing is not encryption; it is authentication (adding MACs) followed by adding chaff.

each contains an English sentence, while the chaff packets contain random bits, then the adversary will have no difficulty in winnowing the wheat from the chaff himself.

On the other hand, if each wheat packet contains a single bit, and there is a chaff packet with the same serial number containing the complementary bit, then the adversary will have a very difficult (essentially impossible) task. Being able to distinguish wheat from chaff would require him to break the MAC algorithm and/or know the secret authentication key used to compute the MACs. With a good MAC algorithm, the adversary's ability to winnow is nonexistent, and the chaffing process provides perfect confidentiality of the message contents. To make this clearer with an example, note that the adversary will see triples of the form:

(1,0,351216)
(1,1,895634)
(2,0,452412)
(2,1,534981)
(3,0,639723)
(3,1,905344)
(4,0,321329)
(4,1,978823)
...

and so on.

I stress that the sending process for chaffing and winnowing is not encryption; it is authentication (adding MACs) followed by adding chaff.

Let us assume that the original message is broken into very short (one-bit) packets, and that MACs have been added to each such packet to create the wheat packets. (There is some obvious inefficiency here, since each wheat packet may end up being, say about 100 bits long, but only transmits one bit. Here each MAC might be 64 bits in length, and each serial number 32 bits long. Additional bits might also be present to identify sender, receiver, etc.)

Such a message sequence is not encrypted, and the process for creating such a message sequence would presumably not be export-controlled, since the message bits are "in the clear" and nicely labelled with serial numbers.

The process of creating chaff is also easy: just create a chaff packet with whatever serial number and packet contents you may like, and include a random 64-bit MAC value. This MAC value is overwhelmingly likely to be bad, and thus the packet created is overwhelmingly likely to be chaff. (The chances of creating a good packet are one in 2^{64} —approximately one in 10^{19} —which is effectively negligible.) The person creating the chaff (the "chaffer") would do so having seen the wheat packets, and would make chaff packets up that have the same serial numbers as the wheat packets do, but with complementary packet contents. Again, it is assumed here that an adversary, not knowing the secret authentication key, can not distinguish a good (wheat) packet from a bad (chaff) one.

It is especially intriguing to now observe that creating chaff does not require knowledge of the secret authentication key! That is, creating chaff is done by creating bogus packets with bogus randomly guessed (and thus bad) MACs; to randomly guess a MAC requires no knowledge of the secret authentication key.

We could thus have the following intriguing scenario: Alice is communicating with Bob using a standard packet-based communication scheme. Each packet is authenticated with a MAC created using a secret authentication key known only to Alice and Bob. (In practice, they might use a different key for packets in each direction, although this is not necessary if the packet contents identify sender and receiver.) Furthermore, each packet happens to contain only a single "message bit." (Alice wrote their software, and it contained a bug that caused this unusual behavior.)

So far, Alice and Bob are not encrypting anything, and are using standard messaging techniques that would not be considered as encryption and that would not be export-controlled. Alice and Bob have no intention of achieving confidentiality of their messages from an eavesdropper.

Now, Alice's packets to Bob may be routed from her computer through the computer of her Internet service provider, run by Charles, on another floor of her building, before being sent on to more major trunks of the Internet and then on to Bob.

Charles' computer, for whatever reason, then adds "chaff" packets to the packet sequence from Alice to

Bob. All of sudden, Charles' activities provide a very high degree of confidentiality for the communications between Alice and Bob! Alice's and Bob's software have not been modified in the least to achieve this confidentiality! Charles does not know the secret authentication key used between Alice and Bob! Alice and Bob did not even want or care to have confidential communications! Charles is not using encryption and does not know any encryption key! Amazing!

Clearly, the cause of the confidentiality is Charles's activities, but Charles has no encryption key or decryption key that he could give to law enforcement. Alice and Bob share an authentication key, but do not perform any encryption, and have no encryption or decryption keys.

Law enforcement may be able to tap the (unencrypted) line from Alice to Charles, but that might be difficult to arrange without Alice's knowledge, as Alice and Charles are in the same building, and may even be friendly or colluding. While Charles' chaffing activities may be suspicious, they don't constitute encryption and don't involve any knowledge of keys on his part; there is no key information he could give to any law enforcement agency.

In a variation on the above scenario, Charles is not "adding chaff" but merely multiplexing the stream of packets from Alice to Bob with another stream of packets (say from David to Elaine). To Bob, the stream of packets from David to Elaine looks like chaff, and is discarded. But to Elaine, the converse holds, and she discards the stream of packets from Alice to Bob as chaff. What is wheat to one pair of communicants is chaff to the other pair, and vice versa. Such a situation could arise where Charles is managing a broadcast channel such as a satellite link; here both parties naturally receive the stream of intermingled packets. If the only way to distinguish one stream from another is by the correctness of the MACs, then an adversary will have a hard time separating the streams. (Of course, if there are exactly two streams being multiplexed, then Alice and Bob can read the stream from David to Elaine, and vice versa.)

In such a scenario, the obvious tack for law enforcement to take would be to demand to have access to

the secret authentication key shared by Alice and Bob. But access to authentication keys is one thing that government has long agreed that they don't want to have. Having such access would allow the government to forge authentic-looking packets for any pair of parties that are communicating. This is way beyond mere access to encrypted communications, as loss of such authentication keys could wreak massive havoc to the structure and integrity of the entire Internet, allow hackers not only to overhear private messages, but to actually control computers, perhaps to shut down power systems or to airline traffic control systems, etc. The power to authenticate is in many cases the power to control, and handing all authentication power to the government is beyond all reason, even if it were for well-motivated law-enforcement reasons; the security risks would be totally unacceptable.

One could imagine that Alice and Bob are merely authenticating their packets to each other, and that it is not Charles but instead a rogue law enforcement agent who is introducing the chaff, and then introducing the authenticated and chaffed message as potential justification to a judge for demanding the authentication key shared by Alice and Bob. If law enforcement had unrestricted right to plaintext, then it could demand surreptitious access to all authentication keys, even when confidentiality techniques were not being used by the participants! Again, such risks are too great to be accepted.

Similarly, a rogue law enforcement agent could introduce the chaff to Alice and Bob's authenticated packet stream, and then attempt to bring Alice and Bob to court for violating some anti-encryption or anti-confidentiality law. How can Alice and Bob defend themselves against this framing attack? They did nothing but send authenticated packets to each other! Again, this shows the difficulty (or impossibility) of drafting any kind of reasonable law restricting encryption or confidentiality technology.

It is possible to make the chaffing and winnowing technique much more efficient, allowing many bits per packet instead of just one. Here is one approach. Suppose Alice has a one-megabit message. She might pre-process the message using an "all-or-nothing" or "package transform" (Rivest 1997)—this is a keyless (non-encryption) transform that takes the

While Charles' chaffing activities may be suspicious, they don't constitute encryption and don't involve any knowledge of keys on his part; there is no key information he could give to any law enforcement agency.

Chaffing and winnowing bear some relationship to steganography.

[...] the adversary may know (or suspect) that there are two different kinds of packets, but he is unable to distinguish them because he does not possess the secret authentication key.

message and produces a “packaged message” with the property that the recipient (Bob) can’t produce the original message unless he has received the entire packaged message. The packaging operation can be undone by anyone who receives the packaged message; as noted, packaging is not encryption and there are no shared secret keys involved in the packaging operation. Alice might want to do so because she wants to ensure that Bob either sees all of the message or none of it; he doesn’t ever see just part of it. Unless the entire packaged message is received, the parts received effectively look like random noise.

Alice then breaks her packaged message into 1024-bit blocks, authenticates each block with a MAC, and transmits the result to Bob. This message is packaged and authenticated, but not encrypted: an eavesdropper can easily reconstruct the message given all of the blocks.

However, Charles can add 1024-bit chaff blocks, where each chaff block has 1024 bits of random data and a random (and presumably wrong) MAC. Again, adding the chaff provides extremely strong confidentiality, since an eavesdropper can not distinguish the chaff from the wheat. Other transforms, besides the packaging transform, might work as well.

For an adversary, the difficulty of separating the wheat blocks from the chaff will be proportional to the number of ways a subsequence of blocks can be picked as and tested for being wheat; this will be exponential in the total number of blocks, assuming that the fraction of chaff blocks is guaranteed not to be close to zero or close to one. We note that when packaging is used, it is not necessary to have as many chaff packets as wheat packets, since the adversary must identify the wheat packets precisely (with no omissions or deletions) in order to retrieve the message. Thus, for long messages, the relative number of chaff packets needed can be quite small, and the extra bandwidth required for transmitting chaff might be insignificant in practice.

Chaffing and winnowing bear some relationship to steganography. I am reminded of the steganographic technique of sending an innocuous-looking letter whose letters are written in two different, but very similar fonts. By erasing all letters in one font, the hidden message written in the other font, remains.

For this technique (as with most steganographic techniques), security rests on the assumption that the adversary will not notice the use of two fonts. With chaffing and winnowing, the adversary may know (or suspect) that there are two different kinds of packets, but he is unable to distinguish them because he does not possess the secret authentication key.

Chaffing and winnowing also bear some resemblance to encryption techniques. Indeed, the process of authenticating packets and then adding chaff achieves confidentiality, and so qualifies as encryption by anyone who uses a definition of encryption that is so broad as to include all techniques for achieving confidentiality. But this fails to note the special structure here, wherein a non-encrypting key-dependent first step (adding authentication) followed by a non-encrypting keyless second step (adding chaff) achieves confidentiality. Since the second step can be performed by anyone (e.g. Charles in our example), and since the first step (adding authentication) may be performed for other good reasons, we see something novel, where strong confidentiality can even be obtained without the knowledge and permission of the original sender. (Variations on chaffing and winnowing, such as omitting the plaintext bits altogether and letting the receiver infer them from the MAC’s, destroy these nice properties.)

I note that the use of MAC’s can be replaced by digital signatures. Not the ordinary kind of digital signatures, since then anyone would be able to distinguish wheat from chaff. But the recent “designated verifier signatures” of Jakobsson, Sako, and Impagliazzo (Jakobsson et al ’96), which can only be verified by those the signer designates, would work fine. (Chaum has also independently invented the same concept.)

I note that it is possible for a stream of packets to contain more than one subsequence of “wheat” packets, in addition to the chaff packets. Each wheat subsequence would be recognized separately using a different authentication key. One interesting consequence of this is that if law enforcement were to demand to see an authentication key so it could identify the wheat, the sender could yield up one such key that identifies a wheat subsequence containing an innocuous message as the wheat, and leaving ev-

everything else as “chaff”. The real message would still be buried in the chaff. This is reminiscent of the technique of “deniable encryption” proposed by Canetti et al. (1997).

In the chaffing and winnow approach, Alice and Bob use standard authentication techniques, and then someone adds chaff to the sequence of authenticated packets. It is worth observing that Alice and Bob can obtain a covert or subliminal channel by replacing a portion of each MAC for an ordinary message by a portion of the ciphertext for a hidden message. Without an authentication key, law enforcement cannot detect this channel. But this is outside our model.

It is also worth noting that the ability to bootstrap from authentication techniques to confidentiality mechanisms is not new. For example, two parties can use authenticated Diffie-Hellman to agree upon an encryption key. In such a case, the parties initially have only each other’s signature verification keys. After the protocol is over, they have a secret shared key that they can use for encryption purposes. Chaffing and winnowing differ in that the two parties involved may not even explicitly take any steps to achieve confidentiality (if someone else is adding the chaff).

Another example of using authentication to achieve confidentiality occurs in baseball—a coach will signal to a runner by giving a sequence of signals, but the real signal is the one immediately following a previously agreed-upon authenticator signal.


A final example of using authentication to achieve confidentiality occurs in the Rex Stout’s novel “The Doorbell Rang.” Two men wish to communicate privately, but fear that the FBI has bugged the room. They agree when the speaker raises a finger, his statements are to be disregarded. Of course, the FBI’s bugs can’t tell if the speaker has his finger raised or lowered!

In summary, we have introduced a new technique for confidentiality, called “chaffing and winnowing”. This technique can provide excellent confidentiality of message contents without involving encryption or steganography. As a consequence of the existence of chaffing and winnowing, one can argue

that attempts by law enforcement to regulate confidentiality by regulating encryption must fail, as confidentiality can be obtained effectively without encryption and even sometimes without the desire for confidentiality by the two communicants. Law enforcement would have to seek access to all authentication keys as well, a truly frightening prospect.

Mandating government access to all communications is not a viable alternative. The cryptography debate should proceed by mutual education and voluntary actions only.

Acknowledgments

Thanks to my dad for suggesting the term “winnowing,” to Mark Lomas for noting that multiplexing two streams may allow each to serve as chaff for the other, and to Peter Wayner for suggesting the relationship to deniable encryption. Thanks to Adi Shamir and David Gifford for suggesting the basic idea underlying the more efficient implementation of chaffing and winnowing; Aaron Gifford first noted that the number of chaff packets might be small in this case. Thanks also to Matt Blaze and Markus Jakobsson for comments on the original write-up. And finally thanks to Bruce Balden and Enzo Michelangeli for bringing the Rex Stout reference to my attention. 

References

- [1] Canetti, Ran, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky, “Deniable Encryption”, Proceedings CRYPTO ’97 (Springer 1997), 90—104. <ftp://theory.lcs.mit.edu/pub/tcryptol/96-02r.ps>
- [2] Jakobsson, Markus, Kazuo Sako, and Russell Impagliazzo, “Designated Verifier Proofs and Their Applications”, Proceedings Eurocrypt ’96 (Springer 1996), 143—154. <http://www.bell-labs.com/user/markusj/dvp.ps>
- [3] Krawczyk, H., Bellare, M., and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication”, RFC2104, February 1997. (Available at <ftp://ds.internic.net/rfc/rfc2104.txt>)
- [4] Rivest, R. “All-Or-Nothing Encryption and the Package Transform,” Proceedings of the 1997 Fast Software Encryption Conference (Springer, 1997). Also on <http://theory.lcs.mit.edu/~rivest/fusion.ps>.
- [5] Stout, Rex. The Doorbell Rang: A Nero Wolfe Novel. (Viking Press, 1965).
- [6] Wayner, Peter. Disappearing Cryptography: Being and Nothingness on the Net. Academic Press, 1996.

“chaffing and winnowing” [...] can provide excellent confidentiality of message contents without involving encryption or steganography.

DES, Triple-DES and AES

Eli Biham

Computer Science Department
Technion — Israel Institute of Technology
Haifa 32000, Israel

Lars R. Knudsen

Department of Informatics
University of Bergen
Hi-techcenter, N-5020 Bergen, Norway

Introduction

The Data Encryption Standard (DES) [21] is by far the most popular block cipher. Around the world, governments, banks, and standards organizations have made DES the basis of secure and authentic communication [24]. DES can be seen as a special implementation of a *Feistel* cipher, named after Horst Feistel [14]. Four modes of operation for block ciphers and in particular for DES were standardized [22]. The four modes are the Electronic Code Book (ECB), the Cipher Block Chaining (CBC), the Cipher Feedback (CFB), and the Output Feedback (OFB). The ECB mode is the native mode, well-suited for encryption of short keys or very short plaintexts. It is not suited for the encryption of larger plaintexts, since equal plaintext blocks are encrypted into equal ciphertext blocks. To avoid this, it is recommended to use the CBC mode, in which each ciphertext block depends of the previous ciphertext blocks, in addition to its plaintext block. In some applications there is a need for encryptions of one character or one bit at a time, instead of full blocks of eight bytes. For that purpose the CFB and OFB modes are suitable. The OFB mode produces a stream of randomly looking data, by encrypting an initial value iteratively, and the ciphertext is the result of exoring this stream to the plaintext.

The Data Encryption Standard (DES) [21] has been the subject of intense debate and cryptanalysis. Immediately after the introduction of DES in the mid 70's, it was criticized for its short key length of 56 bits, and it was argued that *brute-force* attacks (independent of the internal structure of the block cipher) might be applicable. Already in 1977, Diffie

and Hellman proposed a special purpose exhaustive search machine at the cost of about \$20,000,000 which can find a key within a day. In 1993, Wiener [28] designed a machine which costs only about \$1,000,000 and can find a key within 3.5 hours in average. Using today's technology he estimates that such a machine can find a key within 35 minutes in average [29]. More recently, in response to RSA data security challenges, exhaustive search for DES keys was actually applied twice over the Internet within 90 and 40 calendar days [11, 13], confirming that 56-bit keys are too short for current applications.

Since 1990 three successful attacks on DES have been reported, which find the key faster than by an exhaustive search. Such *short-cut* attacks exploit the internal structure of the block cipher. The two most important short-cut attacks on block ciphers are the differential cryptanalysis by Biham and Shamir [7] and the linear cryptanalysis by Matsui [19]. Differential cryptanalysis makes use of so-called *differentials* (A,B), that is, pairs of plaintexts with difference A, which after a certain number of rounds result in a difference B with a high probability. Linear cryptanalysis makes use of so-called *linear hulls*, that is, the parity of a subset of plaintext bits which after a certain number of rounds equals the parity of a subset of ciphertext bits with a probability sufficiently far away from one half. The differentials and hulls can be used to derive information on the secret key. Although, for DES both these attacks are faster than an exhaustive search for the key, the attacks require an unrealistic number of chosen or known plaintexts encrypted under the secret (unknown) key.

There are other attacks on block ciphers which given only the ciphertext can extract information about the plaintext without recovering the key. An example of such an attack is the *matching ciphertext attack* [16, 10], which depends only the block size of the cipher. An attacker collects blocks of ciphertexts and looks for two equal ciphertext blocks. When encrypted in the ECB mode of operation an attacker gets the information that the two plaintext blocks are equal. The attack works also when the ciphertexts are encrypted using the CBC and CFB modes of operation. Consider the CBC mode. Let P_i and C_i denote the plaintext and ciphertext blocks. Then $C_i = E_K(P_i \oplus C_{i-1})$, that is, the current plain-

Eli Biham is interested in the design and analysis of cryptographic algorithms. He can be contacted at biham@cs.technion.ac.il. His WWW page is at <http://www.cs.technion.ac.il/~biham/>.
Lars R. Knudsen is an Associate Professor at the Univeristy of Bergen, Norway. He can be contacted at lars.knudsen@ii.uib.no. His WWW page is at <http://www.ii.uib.no/~larsr/>.

The Data Encryption Standard (DES) [21] has been the subject of intense debate and cryptanalysis. Immediately after the introduction of DES in the mid 70's, it was criticized for its short key length

text block is exclusive-ored with the previous ciphertext block before being encrypted. But if $C_i = C_j$ then since encryption with any key K is invertible, the attacker can deduce that $P_i \oplus C_{i-1} = P_j \oplus C_{j-1}$, and thus that the difference of the two plaintext blocks $P_i \oplus P_j$ is $C_{i-1} \oplus C_{j-1}$. With a block size of n bits, a match $C_i = C_j$ is expected after the encryption of about $2^{n/2}$ blocks. For DES this is only 2^{32} blocks.

Triple-DES and its modes of operation

For a higher level of security against exhaustive key search attacks it is often recommended to use triple-DES [12], which encrypts a plaintext three times with three different keys, or to use two-key triple DES [25], which encrypts a plaintext with a key K_1 , then decrypts with a key K_2 and finally encrypts again with the same key K_1 . This increases the key lengths to 168 or 112 bits respectively. However, the block lengths of 64 bits of these proposals are the same as for DES, and the matching ciphertext attack is still a problem.

For several years the X9.F1 committee of the American National Standards Institute (ANSI) is working on adopting a suite of modes of operation for triple-DES encryption [1]. The proposed modes are based on the four modes of operation of DES, with the underlying cipher replaced by triple DES. Interleaved variants, which encrypt three partial streams containing every third block of the original stream, are also proposed for obtaining better speed in hardware.

One of these proposed modes is the Triple DES Cipher Block Chaining (TCBC) mode, where the feedback block is the ciphertext block (computed by three DES encryptions). This mode is also called the *outer-CBC* mode [15]. This mode is described in Figure 1. However, this mode is vulnerable to the matching ciphertext attack, and is inherently three times slower than DES. Therefore, it has been proposed to use triple-DES in a cipher block chaining mode with internal feedback, called the *inner-CBC* mode [15], where the feedback is applied after each single DES encryption, and therefore can be viewed as applying the CBC mode three times with three different keys. This mode is described in Figure 2. This mode is not vulnerable to the matching ciphertext attack, and was expected to be as secure

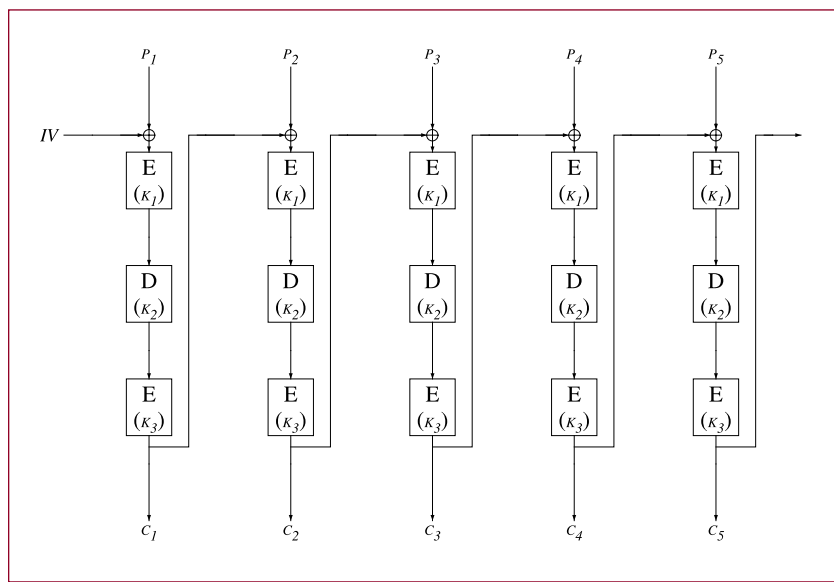


Figure 1: The *outer-CBC* mode (TCBC)

as three-key triple-DES against key recovery attacks. As the best published attack against three-key triple-DES required 2^{112} complexity [20]¹ it was expected that attacks against this mode require more than 2^{112} operations, especially as the attacker is unable to know the internal feedbacks, and therefore, cannot even search for the key given a plaintext/ciphertext pair. Moreover, this mode can be pipelined in hardware and therefore can be applied with the same speed as of single DES.

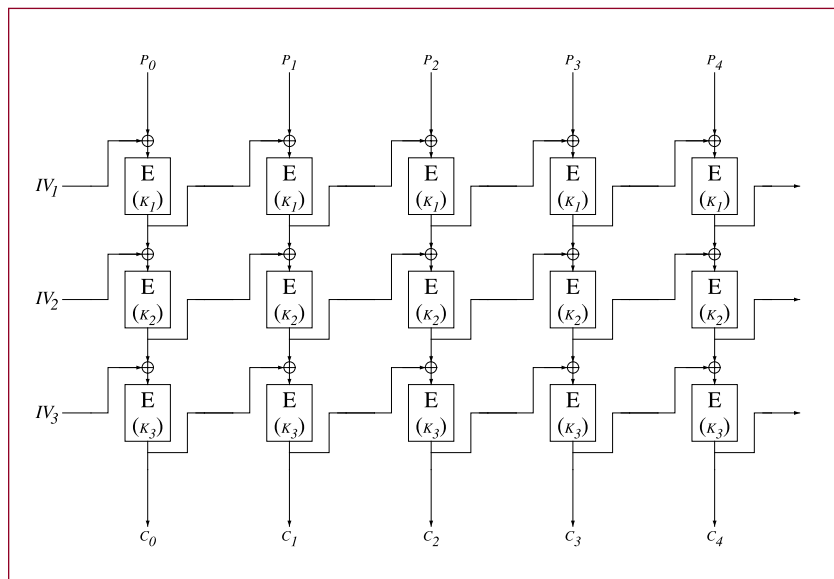


Figure 2: The *inner-CBC* mode

¹ Recently Lucks [18] devised an attack on three-key triple-DES with complexity about 2^{109} .

The CBCM mode was included in the ANSI triple DES modes of operation proposed standard [1], in addition to the modes described earlier. This standard was almost accepted in September 1997

As a consequence of our attack, ANSI decided to remove the CBCM mode from the standard.

However, in a series of papers [3, 4, 6], the first author analyzed a large number of multiple modes of operation, and in particular showed how to mount a key-recovery attack [4] against the inner-CBC mode. The complexity of this attack is considerably smaller than one would expect for triple encryption schemes and only slightly higher than the complexity of attacking single modes. The proposed attacks are based on the ability of the attacker to control the feedbacks, even though he does not know their actual values. In the attack on the inner-CBC mode, the attacker chooses tuples of ciphertexts of the form (C, C, C, C) , where all four blocks are the same. During decryption, the inner-CBC mode ensures that the decrypted data after decryption of one CBC layer becomes of the form $(?, X, X, X)$, where the three X blocks are equal (but unknown to the attacker). Similarly after decryption of two layers the data becomes $(?, ?, Y, Y)$, where the $?$'s are unknown, but the two Y 's are equal, and after decryption of all the three layers the data becomes $(?, ?, ?, Z)$ for some Z . The attack requires 2^{33} such tuples for different values of C . Due to the birthday paradox, it is expected that there is a pair of two tuples, say with values C_1 and C_2 that lead to the same value of X and therefore same values of Y and Z . These C_1 and C_2 can be identified given the plaintext since they have the same Z . Once they are identified, the attacker only has to search exhaustively for one DES key K_3 that satisfies $D_{K_3}(C_1) \oplus C_1 = D_{K_3}(C_2) \oplus C_2$. Given K_3 , it is possible to find the other keys in a similar way. The total complexity is only a few times larger than an exhaustive search of a DES key, rather than being about 2^{112} .

The CBCM mode

In [9, 10] Coppersmith, Johnson, and Matyas propose the *CBC with OFB Masking* (CBCM) mode of operation for triple-DES. The CBCM mode was specially designed to withstand the attacks described in [4, 5], the dictionary attack, and the matching ciphertext attack. The CBCM mode is similar to the CBC mode when two-key triple-DES is used as the underlying cipher, with the addition of mixing with an OFB mask between the first and the second layers, and again with the same mask between the second and the third layers. The mask is the output of an OFB mode using a third key. Figure 3 describes this mode, where E and D denote encryption respectively decryption with the underlying block cipher.

The disadvantage of the proposal is, that it uses four DES encryptions using three different DES keys to encrypt each 64-bit plaintext block. In [9, 10] it is mentioned that the attacks in [4] that use internal feedbacks for the benefit of the attacker leave little hope for devising modes with internal feedbacks. In particular it is mentioned that the inner-CBC mode is weak due to such feedbacks, while on the other hand modes with only outer feedbacks are unsatisfactory. This motivated the design of a more complex mode which has both outer feedbacks, and internal feedbacks, but in which the internal feedbacks may not be controlled by the attacker, as they are the output of an OFB mode. It is claimed in [9] that the CBCM mode is immune against the kind of attacks described in [4].

The CBCM mode was included in the ANSI triple DES modes of operation proposed standard [1], in addition to the modes described earlier. This standard was almost accepted in September 1997, and was delayed only in order to correct some typos found in the proposal. The corrected version had to be finally accepted a few weeks later, and this attack was found just before this final vote. As a consequence of our attack, ANSI decided to remove the CBCM mode from the standard.

The attack on the CBCM mode

In the attack on the CBCM mode, we replace the need to control the feedbacks by making use of their structure. This is done using fixed points of the underlying cipher. A fixed point of a function f is a value x , such that $f(x) = x$. The first main observation in our attack is that when the input to the middle decryption is a fixed point, the middle decryption and the two mixings with the masks are eliminated, and thus the whole triple encryption reduces to a double encryption using the same key twice. When generating sufficiently many plaintext and ciphertext blocks, one has a high probability of encountering one or several encryptions with fixed points. The remaining problem is to find the blocks where this happens. The second observation is that for most functions f , there is exactly one fixed point x . The attack requires a huge number of chosen ciphertext blocks, larger than the period of the OFB stream. We choose 2^{64} ciphertext blocks of the same value C_1 , followed by 2^{64} ciphertext blocks of the some other value C_2 . Given the plaintexts, it is easy

to identify the period of the OFB stream. Now, we try all the 2^{56} candidates K' of the key K_1 , for each we decrypt C_1 and C_2 under the same K' . By comparing the results to the plaintext blocks we find candidate blocks with fixed points in the middle decryption. However, the fixed point is expected to be unique, and thus we identify that the two middle encryptions have the same data. This information allows us to identify relations between two additional blocks with the same mask values but exchanged ciphertexts. These additional blocks are then used to verify whether K' is not K_1 . The full details of the attack appear in [8].

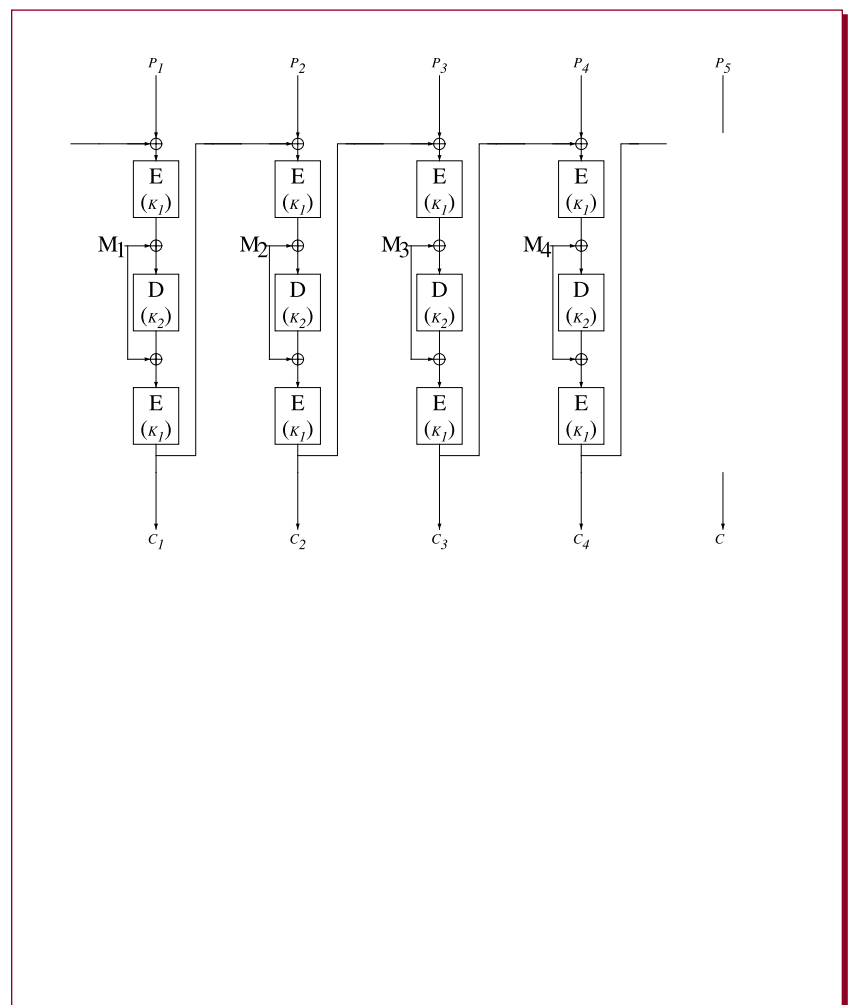
The attack needs about 2^{65} chosen ciphertext blocks, runs in time equivalent to 2^{58} triple-DES encryptions and also requires the memory for keeping the plaintexts. Thus the attack is theoretical, and is not expected to be applied in practice. However, the complexity is several orders of magnitude lower than the claimed complexity of attacking this mode, and is crucially lower than the complexity of the best key-recovery attack on triple DES.

Possible improved solutions

There are several straightforward measures to thwart this attack, by modifying the CBCM mode. However, there is no certainty that these measures actually increase the strength. We believe that changing the key of the last encryption layer to a fourth key K_4 would thwart the attack. On the other hand, we recommend not to change the second OFB mixing to a different stream, as there are attacks on other modes which can recover the two OFB streams in such case [6].

Another possible solution is to use other (hopefully more secure) modes of operation. For example, the first author proposed the OFB[CBC,CBC,CBC⁻¹] mode in [6] (this notation means, xor the plaintext with an OFB stream generated by one DES key, encrypt with the CBC mode with a second key, xor again with the same OFB stream, encrypt again with the CBC mode with a third key, xor once again with the *same* OFB stream, apply CBC mode decryption with a fourth key and finally xor with the *same* OFB stream), with the initial value computed as the result of a message authentication code (MAC) on the transmitted initial value [8]. This mode is still unbroken, and the best known attack has complex-

ity about 2^{112} , even under the most extreme attacking assumptions (such as attacks in which the attacker can get many decrypted chosen ciphertexts with the same chosen initial values [27]). This mode can be pipelined, and thus can be as fast as single DES in hardware, and as fast as CBCM in software.



The solution might also be using DES as a building block of a more secure block cipher. In [17] the second author devised such a block cipher, called *DEAL*, with a conjectured security level of about 2^{120} . The construction is simple. The plaintext blocks of 128 bits are divided into two blocks of 64 bits each; the left half is input to DES using a key K_1 , the output is exclusive-ored to the right half, and the two halves are swapped. The ciphertext is the output of the sixth such round, where in each round a new DES key is used. In other words, DEAL is a six round Feistel cipher using DES in the round func-

- distributed.net/des/.
- [14] H. Feistel, *Cryptography and computer privacy*, Scientific American, 228(5):15-23, 1973.
 - [15] B. S. Kaliski and M. J. B. Robshaw, *Multiple encryption: Weighing security and performance*, Dr. Dobbs Journal, pp. 123-127, January 1996.
 - [16] L.R. Knudsen, *Block Ciphers — Analysis, Design and Applications*, PhD thesis, Aarhus University, Denmark, 1994.
 - [17] L. R. Knudsen, *DEAL — A 128-bit Block Cipher*, Technical Report 151, Department of Informatics, University of Bergen, Norway, February 1998.
 - [18] Stefan Lucks, *Attacking Triple Encryption*, proceedings of Fast Software Encryption, Paris, Lecture Notes in Computer Science, pp. 239-253, 1998.
 - [19] Mitsuru Matsui, *Linear Cryptanalysis Method for DES Cipher*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'93, pp. 386-397, 1993.
 - [20] R. C. Merkle, M. E. Hellman, *On the Security of Multiple Encryption*, Communications of the ACM, Vol. 24, No. 7, pp. 465-467, July 1981.
 - [21] National Bureau of Standards, *Data Encryption Standard*, U.S. Department of Commerce, FIPS pub. 46, January 1977.
 - [22] National Bureau of Standards, *DES Modes of Operation*, U.S. Department of Commerce, FIPS pub. 81, December 1980.
 - [23] National Institute of Standards and Technology, *Advanced Encryption Standard (AES) Development Effort*, http://csrc.nist.gov/encryption/aes/aes_home.htm.
 - [24] M.E. Smid and D.K. Branstad, *The {Data Encryption Standard}: Past and future*, In G.J. Simmons, editor, *Contemporary Cryptology — The Science of Information Integrity*, chapter 1, pages 43-64. IEEE Press, 1992.
 - [25] W. Tuchman, *Hellman presents no shortcut solutions to DES*, *IEEE Spectrum*, 16(7):40-41, July 1979.
 - [26] Paul C. van Oorschot, Michael J. Wiener, *A known-plaintext attack on two-key triple encryption*, *Advances in Cryptology*, proceedings of EUROCRYPT'90, LNCS 473, pp. 318-325, 1990.
 - [27] David Wagner, *Cryptanalysis of Some Recently-Proposed Multiple Modes of Operation*, proceedings of Fast Software Encryption, Paris, Lecture Notes in Computer Science, pp. 254-269, 1998.
 - [28] Michael J. Wiener, *Efficient DES Key Search*, presented at the Rump session of CRYPTO'93. Reprinted in *Practical Cryptography for Data Internetworks*, W. Stallings, editor, IEEE Computer Society Press, pp. 31-79, 1996.
 - [29] Michael J. Wiener, *Efficient DES Key Search — An Update*, *CryptoBytes*, Vol. 3, No. 2, pp. 6-8, 1998.


DES-II Challenges Solved

On January 13, 1998 the first of two recent DES challenges was offered to the cryptographic community. While the total prize money for each of the DES-II challenges is \$10,000, the full amount is paid only when the time required to find the solution is less than or equal to 25% of the time required for the previous challenge. The prize drops to \$5,000 when the time for solution is less than or equal to 50% of the previous time and \$1,000 is paid for a time within 75% of that mark.

The organization *distributed.net* used the idle time of computers throughout the world to solve the first DES-II challenge of 1998. They coordinated the efforts of 22,000 participants and linked together over 50,000 CPUs. The team searched over 61 quadrillion, 254 trillion keys at a peak rate of 26 trillion keys per second. After searching through 85% of the keyspace, the winning key was found which revealed the hidden message "Many hands make light work." While the team had to search nearly the

entire keyspace before finding the key, the calendar time for solving the contest was 40 days thereby netting a reward of \$5,000.

The second challenge was launched on July 13, 1998 and was solved in under three days by the Electronic Frontier Foundation (EFF). EFF was well within the ten calendar day mark determined by *distributed.net*'s solution of the previous challenge, and hence, EFF received the full \$10,000 prize. EFF used a single machine called the EFF DES Cracker to win the contest. The Cracker was built for less than \$250,000, and is able to search through 88 billion keys per second. It took the Cracker 56 hours to find the key with which to reveal the hidden message, "It's time for those 128-, 192-, and 256-bit keys."

More information on these and other challenges maintained by RSA Laboratories can be found at <http://www.rsa.com/rsalabs/html/challenges.html>. 

A N N O U N C E M E N T S

In this issue:

- ***Performance
Comparison of
Public-Key
Cryptosystems***
- ***Smart Card
Crypto-
Coproductors for
Public-Key
Cryptography***
- ***Chaffing and
Winnowing:
Confidentiality
without Encryption***
- ***DES, Triple-DES
and AES***

***For contact and
distribution
information, see
page 2 of this
newsletter.***



RSA Laboratories.

A Division of RSA Data Security

2955 Campus Drive, Suite 400
San Mateo, CA. 94403-2507

Tel 650/295-7600
Fax 650/295-7599

rsa-labs@rsa.com
<http://www.rsa.com/rsalabs>