VOLUME 1, NUMBER 3 — AUTUMN 1995

RSA LABORATORIES'

CryptoBytes

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

Contents

RSA for Paranoids

Adi Shamir
Applied Math Department
The Weizmann Institute of Science
Rehovot 76100, Israel

One of the most important decisions in practical implementations of the RSA cryptosystem is the choice of modulus size. It is clear that the standard size of 512 bits no longer provides adequate protection, and should be substantially increased. However, the time complexity of modular exponentiation grows rapidly with the size of the modulus, and thus it is difficult to choose a size which combines efficient operation with long term security. In this note we describe a new variant of the RSA cryptosystem called "unbalanced RSA", which makes it possible to increase the modulus size from 500 bits to 5,000 bits without any speed penalty.

Conventional RSA

The security of the RSA cryptosystem depends (but is not provably equivalent to) the difficulty of factoring the modulus n, which is the product of two equal size primes p and q. Until recently, the minimum recommended size of n was 512 bits. However, recent advances in factoring algorithms make it necessary to increase this size. Since the time complexity of RSA computations is already substantial, and grows cubically with the size of the modulus, the new size should by necessity be a compromise between efficiency and security. The choice is particularly difficult for paranoid organizations whose encrypted messages should remain secret for several decades, since it is almost impossible to predict the progress of factoring algorithms over such a long period of time. The only reasonable course of action is to use huge margins of safety, but this will make the RSA operations extremely slow.

All the known factoring algorithms can be divided into two broad types: algorithms whose running time depends on the size of the factors, and algorithms whose running time depends only on the size of the factored number n. The oldest factoring algorithms typically searched for the smallest factor p of n, and were thus of the first type. However, modern algorithms tend to use indirect approaches which require the same time to find a single digit or a fifty digit prime factor of n.

The fastest factoring algorithm of the first type is currently the elliptic curve method. Its asymptotic running time is $\exp(O((\ln(p))^{1/2} \cdot (\ln\ln(p))^{1/2}))$, but its basic operations are very slow. The largest factor ever found in practice with this algorithm was about 145 bits long (A. Lenstra, private communication), and it is very unlikely that this algorithm will be able to find the 256 bit factors of 512 bit RSA keys in the next few years.

Factoring algorithms of the second type are much faster, since they can use a wider array of mathematical techniques. The best algorithm of this type is currently the general number field sieve. It has an asymptotic complexity of $\exp(O((\ln(n))^{1/3} \cdot (\ln\ln(n))^{2/3}))$, and is believed to be capable of factoring 512 bit moduli in 10,000 to 15,000 MIPS-years (A. Lenstra, private communication). This is

(continued on page 3)

Adi Shamir is professor at the Applied Math Department of the Weizmann Institute of Science, Israel. He is a co-inventor of the RSA cryptosystem and can be contacted at shamir@wisdom.weizmann.ac.il.

RSA LABORATORIES
CRYPTOGRAPHIC RESEARCH AND CONSULTATION

Editor's Note

In this, the autumn edition of *CryptoBytes*, we have decided to concentrate on issues related to the use of the RSA cryptosystem.

The main article in this issue is entitled *The Secure Use of RSA*. Adapted from a seminar presented by Ron Rivest at the 1995 RSA Laboratories Seminar Series, this article considers the myths and realities of using the RSA cryptosystem. From low exponent attacks to padding rules, this article provides the ground rules for avoiding a variety of potential attacks on RSA when used either to provide message encryption or to digitally sign documents.

We also have a variety of reports on several recent research results.

At most conferences, time is set aside for researchers to present their latest results outside of the main program.... This means that it is often hard to track down the essential details of what was said.

At most conferences, time is set aside for researchers to present their latest results outside of the main program, and consequently, outside of the conference proceedings. This means that it is often hard to track down the essential details of what was said. As initial steps to make information on rump session items more widely available, we present reports on three items that were presented at recent rump sessions.

A novel idea by Adi Shamir, leading to a rather unconventional implementation of RSA, has some very intriguing properties. In this issue of *CryptoBytes* we include an article by Adi Shamir which provides more details on ideas that were originally presented at the rump session of Eurocrypt '95 earlier this year.

We also have news of two items from the rump session of the Crypto '95 conference. We report on a new "protocol failure" when RSA is used with exponent three and also on a new attack on the envelope method of using a hash function for message authentication. (In one way or another, message authentication with hash functions has featured in all issues of *CryptoBytes* so far!) Also, as part of our ongoing features, we provide an Algorithms Update on some very exciting new work in the analysis of hash functions by Hans Dobbertin.

Finally, in this issue, we introduce a new column: *Frequently Asked Questions*. Extracted from the ongoing enhancements to RSA Laboratories' *Answers to Frequently Asked Questions*, this column will feature answers to new questions on today's cryptography that are of particular interest to our readers. In this issue, the question "How do digital time-stamps support digital signatures?" is answered by Stuart Haber, Burt Kaliski and Scott Stornetta.

The future success of *CryptoBytes* depends on input from all sectors of the cryptographic community and as usual we would very much like to thank the writers who have contributed to this third issue. We encourage any readers with comments, opposite opinions, suggestions, proposals for questions in our *Frequently Asked Questions* column or proposals for items in future issues to contact the *CryptoBytes* editor by any of the methods given below.

Contact and Subscription Information

CryptoBytes will be available by subscription and individual annual subscription to four issues is U.S.$90. To subscribe, contact RSA Laboratories at:

RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065
415/595-7703
415/595-4126 (fax)
rsa-labs@rsa.com

We are hoping that electronic subscription to *CryptoBytes*, via the World-Wide Web, will be available soon. All back issues are available via the World-Wide Web at *http://www.rsa.com/rsalabs/cryptobytes/*.

The *CryptoBytes* editor can be contacted by any of the above methods, or by E-mail to *bytes-ed@rsa.com*.

About RSA Laboratories

RSA Laboratories is the research and development division of RSA Data Security, Inc., the company founded by the inventors of the RSA public-key cryptosystem. RSA Laboratories reviews, designs and implements secure and efficient cryptosystems of all kinds. Its clients include government agencies, telecommunications companies, computer manufacturers, software developers, cable TV broadcasters, interactive video manufacturers, and satellite broadcast companies, among others.

RSA for Paranoids

Continued from page 1

only 2-3 times harder than the factorization of RSA-129 achieved (by a different algorithm) in early 1994, and is likely to be demonstrated within a year.

Since the inception of the RSA cryptosystem, all the record breaking factorizations of RSA keys were based on algorithms of the second type, and it is reasonable to assume that this trend will continue in the foreseeable future. If this is the case, there is no need to increase the sizes of n and its prime factors at the same rate. In this paper we use this observation in order to propose a new variant of the RSA cryptosystem, which we call "unbalanced RSA". Its goal is to provide much higher security at essentially no extra cost.

Unbalanced RSA

As a typical example of the new RSA variant, we describe an "ultimate security" implementation, which is likely to provide long term security even to professional paranoids. In this version we increase the overall size of n by a factor of 10 (to 5,000 bits), and the size of its prime factor p by a factor of 2 (to 500 bits). The size of the other factor q of n is 4,500 bits, and the name of this variant reflects the unequal sizes of the two prime factors of n.

Such a modulus is way out of reach of today's factoring algorithms, and is likely to remain secure even if factoring algorithms of the first type will become much better, and factoring algorithms of the second type will become enormously better. Only a major breakthrough in factoring (such as a practical polynomial time factoring algorithm, which will completely kill the RSA scheme) can possibly threaten it.

The main problem in using such a modulus n is the fact that standard RSA implementations become about 1000 times slower: A typical 500 bit exponentiation which required 1 second on a microprocessor would now require 16 minutes, which is unacceptable.

Since RSA encryption is typically used only in order to exchange session keys for fast secret key cryptosystems, the cleartexts are usually quite short: even three independent keys for triple DES require only 168 bits, and it is unlikely that anyone would use the RSA cryptosystem to exchange secret keys which are larger than p. We can thus assume that the cleartext is in the range $[0, p)$, and that it is padded with random bits to make sure that it is not much shorter than p.

A well known way of speeding up the RSA encryption process $c = m^e \pmod n$ is to use a small encryption exponent e. Since m is only one tenth the size of n, encryption exponents which are smaller than 10 do not provide any wraparound during the modular reduction, and should be avoided. When e is around 20, the size of m^e is about twice the size of n, and thus the wraparound effect is similar to the squaring operation of full size numbers in Rabin's scheme. Note that for such an e, most of the multiplications in the computation of $m^e \pmod n$ are non modular multiplications of relatively small numbers, and only the last squaring is likely to be a full size modular multiplication. The recommended range of e is 20-100, which makes the encryption time with a 5,000 bit modulus comparable to the decryption time with a 500 bit modulus (i.e., less than a second).

The other operation we have to consider is the decryption process $m = c^d \pmod n$ in which c, n and d are 5,000 bit numbers. If we use the Chinese remainder theorem, we can easily compute $m_1 = c^d \pmod p$ via a 500 bit exponentiation, by reducing c modulo p and d modulo $p-1$ at the beginning of the computation. However, we then have to carry out the 4,500 bit exponentiation $m_2 = c^d \pmod q$, which is $9^3 = 729$ times slower.

The main purpose of this note is to show that there is no need to carry out this expensive computation. By definition, m_1 is equal to $m \pmod p$. However, the cleartext m is known to be smaller than p, and thus $m \pmod p$ is simply m itself. By combining these observations, we conclude that m_1 is the original cleartext m, and thus it is just a waste of time to carry out the computation of m_2 modulo q, which will yield the same result.

Other optimizations

While the unbalanced RSA variant has about the same time complexity as the original variant, its space complexity and public key size are 10 times larger. We now show how to use additional optimization techniques to avoid these extra overheads.

Consider first the issue of RAM size. On personal computers the 10-fold increase in the size of the registers is meaningless, but smart cards contain very small RAMs and their cost is directly proportional to the size of their memory. In addition, many smart card manufacturers have already designed RSA

In this paper ... [we] propose a new variant of the RSA cryptosystem, which we call "unbalanced RSA". Its goal is to provide much higher security at essentially no extra cost.

coprocessors which can handle 512 bit moduli, and would be reluctant to redesign them in order to accommodate much larger moduli. However, in most applications the smart card is talking to a powerful PC or workstation, and we can choose to implement either the encryption or the decryption side of the RSA key exchange on the smart card. It turns out to be easier to let the powerful machine choose and encrypt the session key. The arbitrarily long ciphertext is sent to the smart card in a bit serial mode, and the smart card reduces it on-the-fly modulo p by clocking it through the 512 bit input register of its coprocessor. The resultant value is then exponentiated by the 512 bit coprocessor to obtain the session key. The advantage of this approach is that today's coprocessors would be able to handle variable length moduli (dictated by the changing state of the art of factoring algorithms) without expensive periodic replacements of millions of smart cards in large-scale consumer applications.

Another potential drawback of the new variant is the 10-fold increase in the size of the public key directory. However, the unbalanced construction of the new public keys makes it possible to eliminate this problem. Let G be a publicly known pseudo random bit generator which maps each user identity u (name, email address, etc.) into a unique 5,000 bit pseudo random target value t. Each user picks a random 500-bit prime p, and defines the other prime q as a random prime number in the range $[a, a + 2^{50}]$ where a is the smallest integer greater than or equal to t/p. Since the prime numbers are relatively dense, such a q almost certainly exists, and $n = pq$ is very close to the target value t (the difference $s = n - t$ is expected to be about 550 bits long). Each user u publishes this 550-bit s as his public key, and anyone can recover the user's 5,000-bit modulus n by computing $G(u) + s$.

It is important to note that this approach does not make n easy to factor: the only difference between our n and standard n is that we start the search for q at a point which is the ratio of a random number and a prime number, rather than at a random number. Since the distribution of prime numbers p is not completely uniform, the choice of a is not completely uniform. We smooth this slight nonuniformity by specifying a fairly large range of size 2^{50} in which q can be chosen, which makes the exact location of the range's endpoints irrelevant. Once n is chosen, it makes no difference whether we specify it by its bits or by its distance s from another number t, and the fact that t is a pseudo random rather than a truly random number cannot change this fact.

Finally, we note that the technique described in this paper can also be used to double the speed and halve the key size of standard implementations of the RSA cryptosystem in which p and q have similar sizes. The optimization does not apply to the RSA signature scheme, since the verifier cannot carry out computations modulo the unknown p.

... the technique described in this paper can also be used to double the speed and halve the key size of standard implementations of the RSA cryptosystem in which p and q have similar sizes.

ALGORITHMS UPDATE

Collisions in MD4

Recent work by Hans Dobbertin has discovered that collisions for MD4 can be found within a few minutes on a typical PC. Even more impressive is the fact that collisions can be constructed, in around an hour, so that the text they represent makes sense. For an example, see how Alf swindles Ann opposite.

MD4 was among the first of a new generation of what are termed *dedicated hash functions*, designed from first principles to be hash functions rather than being based around some other primitive such as a block cipher. MD5, RIPEMD, the secure hash algorithm SHA and its more recent revision SHA-1 follow the same design approach that was pioneered by Rivest with MD4. Rivest also designed an extended version of MD4 with a 256-bit rather than 128-bit hash value which was considered to be much more secure than MD4. It appears, however, that these new techniques will also compromise extended-MD4.

While MD4 remained secure until now, it was felt that MD4 made little allowance for potential cryptanalytic developments. Consequently, the use of MD4 has been discouraged for some considerable time, and instead the usual recommendation was to replace MD4 with the more conservatively designed

MD5. Dobbertin's work clearly confirms that if MD4 is still being used anywhere, then it should be immediately replaced.

As for the applicability of Dobbertin's techniques to other hash functions such as MD5, RIPEMD and SHA-1, the task facing the cryptanalyst is much more involved. Initial review of those algorithms suggests a limited opportunity for new developments (except on reduced-round versions of RIPEMD which have already succumbed to this type of analysis) and *CryptoBytes* will report on any further developments.

Alf Swindles Ann

Hans Dobbertin
German Information Security Agency
P.O. Box 20 10 63,
D-53133 Bonn, Germany

Alf wanted to sell Ann his house, and Ann was interested. They agreed on a price of $176,495. Alf asked Ann to sign a contract using a digital signature scheme which is based on some public-key algorithm and the hash function MD4. The contract read as follows:

> **********
> CONTRACT
>
> At the price of $176,495 Alf Blowfish sells his house to Ann Bonafide. ...

"The first 20 bytes (each of them is represented by an asterisk above) are random. They have been placed before the text for security reasons!" claimed Alf, and Ann signed the contract. Later, however, Alf substituted the contract file by another which read as follows:

> **********
> CONTRACT
>
> At the price of $276,495 Alf Blowfish sells his house to Ann Bonafide. ...

The contract had been prepared by him such that replacing $176,495 by $276,495 does not change the MD4 hash value!

How Alf did it

For the precise definition of the above digital contract note that the first sixteen 32-bit words are:

Professor Hans Dobbertin is particularly interested in the evaluation and design of cryptographic algorithms. He can be contacted at dobbertin@skom.rhein.de.

M_0 = 0x9074449b
M_1 = 0x1089fc26
M_2 = 0x8bf37fa2
M_3 = 0x1d630daf
M_4 = 0x63247e24
M_5 = 0x4e4f430a
M_6 = 0x43415254
M_7 = 0x410a0a54
M_8 = 0x68742074
M_9 = 0x72702065
M_{10} = 0x20656369
M_{11} = 0x2420666f
M_{12} = 0x2C363731
M_{13} = 0x20353934
M_{14} = 0x20666c41
M_{15} = 0x776f6C42

The twenty bytes of M_0-M_4 are the above mentioned "random bytes". The bytes of M_5, in reverse ordering (according to the definition of MD4) and interpreted as ASCII read as follows:

0a 43 4f 4e = *Line-feed* 'CON',

and so on to M_{15} which reads

42 6c 6f 77 = 'Blow'.

The sequence M_i (i<16) has been chosen such that setting $M'_{12} = M_{12}+1$ and $M'_i = M_i$ for i<16, $i \neq 12$ gives a collision, i.e.

compress(IV; M) = compress(IV; M')

for the compress function of MD4 and its fixed initial value IV. In [1] the basic method of generating such collisions was described. They can be found in less than one hour on a PC. Interpreting M_{12} = 0x2c363731 and M'_{12} = 0x2c363732 we get:

M_{12} = 31 37 36 2c = '176,'
M'_{12} = 32 37 36 2c = '276,'

In view of the definition of MD4 as the iterative application of **compress**, we obtain a collision by taking any bit string and appending it to M and M'.

Where MD4 is still in use, it should be replaced!

Where MD4 is still in use, it should be replaced!

References

[1] Dobbertin, H.: *Cryptanalysis of MD4*, preprint.

More Developments with Keyed Hash Functions

At the rump session of Crypto'95, Bart Preneel in joint work with Paul van Oorschot described a new attack on one method of message authentication using a keyed hash function (see *CryptoBytes* volume 1, number 1).

... the new attack does demonstrate a certificational weakness in the envelope method ...

In what is sometimes called the *envelope method*, a hash function H can be used to provide authentication of a message m under the action of two keys k_1 and k_2 by using the result of $H(k_1 \cdot m \cdot k_2)$ as a message authentication code for the message m. A variety of padding schemes might also be specified as additional security measures.

Previously (see *CryptoBytes* volume 1, number 2), Preneel and van Oorschot have reported that the envelope method of keyed-MD5 allows a MAC forgery attack with "2^{64} known message-MAC pairs and a single chosen text." These known messages have to be the same length and additional reductions in the data requirements are possible if the messages are known to have a particular form.

At the rump session of Crypto '95, Preneel described a new attack on the envelope method which allows recovery of the secret key rather than just a MAC forgery. For the new attack to succeed, the length of the messages being authenticated is carefully chosen by the cryptanalyst to ensure that the action of the second key k_2 is split into two parts, one under the influence of k_a and the other under k_b, where k_2 is equal to the concatenation $k_a \cdot k_b$.

The attack requires two phases, the first of which is in effect the previously mentioned MAC-forgery attack. Using information obtained from this first phase, the cryptanalyst requests the MACs on $2^{|k_a|}$ pairs of chosen messages ($2^{|k_a|+1}$ messages in total) where $|k_a|$ denotes the number of bits in k_a. These new message pairs are chosen according to different guesses for k_a.

When the guess for k_a is correct, the MACs generated for that pair of messages will be the same and k_a can be correctly identified. The rest of k_2 can be found either by exhaustive search, or by using messages of a second carefully chosen length to split the unknown remainder of k_2 once again.

The increased work and data requirements for the new attack over that offering MAC-forgery depend on the length of k_a since this determines the number of guesses a cryptanalyst is required to try in the second phase. However, since the cryptanalyst can choose this length, the requirements of the attack are effectively dominated by the requirements for the MAC-forgery attack.

Since the MAC-forgery attack is barely practical the same must be said of this new attack which recovers the key. However the new attack does demonstrate a certificational weakness in the envelope method since the secret key can be recovered with much less work than the length of the secret key might imply. While the use of additional padding (to prevent the splitting of k_2) would thwart this attack, it seems fair to recommend different mechanisms than the envelope method for providing message authentication with a keyed hash function. *CryptoBytes* will report on future developments.

A Linear Protocol Failure for RSA With Exponent Three

At the rump session of Crypto '95, Matthew Franklin and Michael Reiter of AT&T Bell Laboratories presented a new weakness of RSA with low encrypting exponent. This weakness depends on two messages being encrypted with exponent three with respect to the same RSA modulus, and on a (non-trivial) linear relationship being known to exist between the two messages. In such cases the messages can be computed efficiently by an attacker that has access only to the public key, the two ciphertexts, and the linear relation. This differs from the "small exponent" protocol failure, generalized by Hastad (described opposite in *The Secure Use of RSA*), which requires that messages be encrypted with respect to *different moduli*. It also differs from the "common modulus" protocol failure, observed by Simmons, which requires that a single message be encrypted with *different exponents*. This weakness can be exploited to yield passive attacks on protocols that encrypt related messages. Examples of such protocols are a signature sharing protocol for RSA, proposed by Franklin and Reiter at Eurocrypt '95, and a key distribution protocol, proposed by Tatebayashi, Matsuzaki, and Newman at Crypto '89. This weakness does not affect the encryption of arbitrary unrelated messages.

The Secure Use of RSA

Burt Kaliski and **Matt Robshaw**
RSA Laboratories
100 Marine Parkway, Suite 500
Redwood City, CA 94065-1031

Since its invention over fifteen years ago, the RSA cryptosystem [21] has been submitted to extensive scrutiny and analysis. Its continuing resistance to attack has resulted in its being well trusted and used widely.

One of the most notable features about RSA is its apparent simplicity and considerable elegance. This elegance is due in most part to the algebraic framework in which the algorithm is defined. But this framework can itself be a dangerous environment in which to practice cryptography. Indeed it is straightforward to write down a few mathematical identities that at first sight seem to threaten the integrity of RSA when used either for encryption or for the provision of digital signatures.

As we will see, however, for each of these apparent threats to RSA there is a simple and practical countermeasure. It is all a matter of using the technology correctly. In this article we will show how RSA can be used to its full potential.

The RSA Cryptosystem

RSA is a public-key cryptosystem and digital signature scheme that was invented in the late 1970s at MIT by Ron Rivest, Adi Shamir and Len Adleman [21]. The system depends on modular exponentiation and is defined as follows.

We will denote the public key by (n, e) and the private key by (n, d). The integer n, which is known as the modulus, is chosen as the product of two primes p and q. The integers e and d are exponents and they are chosen so that $ed \equiv 1 (\mathrm{mod}\ \phi(n))$ where $\phi(n) = (p-1)(q-1)$.

Encryption and decryption of a message m can be described as follows. Using the public key (n, e) the sender of the message computes $c = m^e \bmod n$ and c is the encrypted message. The legitimate receiver uses the private decryption key (n, d) to compute $c^d \bmod n$. The underlying mathematics guarantees that

$$c^d \equiv (m^e)^d \equiv m^{(ed)} \equiv m^1 \equiv m \pmod{n}$$

and the original message can be recovered.

Closely following the original Diffie-Hellman model for a public-key cryptosystem [8], the private and public keys in RSA can be used to provide a digital signature. Here the digital signature is generated using the private key (which is known to only one user) and verification of the signature can be completed with the corresponding public key. The signature s of message m with the private key (n, d) is given by $s = m^d \bmod n$ while public verification of the signature s can be achieved using (n, e) since the comparison

$$m \stackrel{?}{\equiv} s^e \pmod{n}$$

will hold for a legitimate digital signature and fail otherwise.

Both encryption and the creation of digital signatures are simple and straightforward operations. The mathematical structure underpinning the cryptosystem ensures that the link between these operations and their reverse can be easily established. But other links can be made and as motivation for what follows we will make note of two, easily established identities:

$$(m_1 * m_2)^e \equiv m_1^e * m_2^e \pmod{n} \quad \text{and}$$
$$(m * r^e)^d \equiv m^d * r \pmod{n}.$$

We will see that exploitation of these identities and other properties can lead to attacks on both the generation of RSA signatures and on RSA encryption. In this article we will describe these attacks and some straightforward and practical counter-measures.

... for each of these apparent threats to RSA there is a simple and practical countermeasure. It is all a matter of using the technology correctly.

Aims of the Attacker

Three consequences of a successful attack on a public-key cryptosystem or digital signature scheme such as RSA might be:

1. recovery of the private key,
2. message decryption, and
3. signature forgery.

Burt Kaliski is chief scientist and Matt Robshaw is a research scientist at RSA Laboratories. They can be contacted at burt@rsa.com or matt@rsa.com.

It is generally accepted that RSA numbers composed of primes of about the same size are among the hardest to factor...

It may well be possible that an attack achieving the second or third result can be mounted without the attacker actually recovering the private key. In this article we consider attacks that might be used to achieve each of these goals in turn, and give rules of thumb which prevent the attacks being successful.

We note that in a practical implementation of RSA there are additional security considerations beyond those covered here. For instance, the interchangeability of signatures and decryption might lead to unexpected security weaknesses if the same key were used for both. Additionally, the source of randomness for key generation and the storage of the private key are vital issues in the secure implementation of RSA. Suffice it to say that no implementation of RSA can be considered fully secure unless all issues, including the system-dependent factors, are taken into account.

Recovering the Private Key

As a first line of attack, an adversary might take the public key (n, e) of some user and attempt to recover the private decrypting (or equivalently the signing) exponent d. Interestingly, one of the links that has been established between the RSA cryptosystem and the difficulty of factoring is based on a number-theoretic result predating RSA [16], which can be used to demonstrate that obtaining a private exponent d from the public exponent e and the modulus n is as hard as actually factoring n. In essence, knowledge of the exponents e and d allows the attacker to factor the modulus n.

... strong primes were introduced so that [...] the modulus n resists the special-purpose factoring methods. The introduction of the elliptic curve method of factoring changed all this.

With this connection in mind, we first consider the most well known (and currently the best) attack that an adversary might mount on RSA when trying to recover the private key, namely an attempt to factor the RSA modulus n.

Factoring

The problem of factoring and its continuing improvements are the topic of a great many articles and papers and we will not attempt to replicate the information that can be found in three, splendid articles [2,17,20].

It is generally accepted that RSA numbers composed of primes p and q of about the same size are among the hardest to factor for their size. (See, however, the article by Adi Shamir in this issue of *CryptoBytes* on using RSA moduli with two primes that are widely differing in size!) For large enough numbers, and certainly for the size of numbers we use in today's implementations of RSA, the *general number field sieve (GNFS)* is the best "general-purpose" factoring method. The older *quadratic sieve* and its variants are faster below a certain size of modulus (currently around 116 decimal digits in length [2]). Most importantly however, users of RSA can determine the current level of factoring ability and make allowances for a certain amount of future improvement. As a consequence, general-purpose factoring need not be a concern to users of RSA if the primes (and hence the RSA modulus) are chosen to be sufficiently large.

Depending on the form of the primes p and q that are multiplied together to give the user's modulus, it might be argued that "special-purpose" factoring methods such as *Pollard's $p-1$ method* [18] and *superencryption attacks* [23], which we discuss next, might end up being faster than using the GNFS. With these and similar threats in mind, *strong primes* were introduced so that p and q were chosen to satisfy a variety of conditions; for instance p might be chosen so that $p-1$ has a large factor r and $r-1$ has a large factor, etc. These conditions, and similar ones on the form of the numbers $p+1$ and $q+1$ would guarantee that the modulus n resists the special-purpose factoring methods.

The introduction of the *elliptic curve method (ECM)* of factoring changed all this [15]. This factoring method has some probability of success regardless of the actual form of the prime. As a consequence, the dominant property for the security of the primes we use is size. In protecting against the ECM factoring technique one protects, with a very high probability, against all special-purpose factoring techniques. In short, large primes are more important than strong primes.

One early proposed attack against RSA is called *superencryption*. Simmons and Norris [23] observed that after a sufficient number of repeated encryptions, the original message would eventually be recovered. This would lead to an attack on RSA if the number of encryptions required were small. But this is not the case if the primes are large and chosen at random and so while an interesting observation, superencryption is not a practical attack. Likewise, the

observation that p and q being very close together in numerical value allows the efficient factorization of n [25], is not relevant to practical security provided p and q are sufficiently large and chosen at random.

Wiener's Attack

There is an interesting attack due to Wiener [24] which allows an attacker to recover the private exponent in an ingenious way. Wiener shows that a continued fraction approximation using the publicly available parameters e and n can provide sufficient information to recover d. This attack will be efficient and practical when the private exponent d is small relative to the RSA modulus; that is when $d < n^{1/4}$.

This attack is a concern if the *private* exponent d is deliberately chosen to be small, perhaps in an attempt to improve the efficiency of decryption or signing. In general use however, it is unlikely that a small private exponent will be generated if the public exponent e is chosen first; when e is small, d is always large enough to resist this particular attack and if e is chosen at random, then with an overwhelming probability, d will be large enough to resist Wiener's attack. If the private exponent d is chosen, it should not be too small.

Using Probabilistic Primes

One frequent question is how the security of RSA might be affected if one of the primes used to compute the modulus is, in fact, not a prime number after all.

First, the factors of n will be smaller than expected, and so it may be easier to factor the modulus with special-purpose factoring methods. Second, except in the unlikely event that we have mistakenly generated a so-called "Carmichael number," decryption and verification will yield incorrect results for most messages and signatures, an occurrence which could be used to reveal the factors of the modulus.

Though a theoretical concern, the possibility of generating a non-prime is not really a practical issue. With modern prime generation methods, the probability that an output is not prime can be made arbitrarily small or even eliminated entirely.

Decrypting Messages

While recovering the private exponent d seems to be difficult provided we generate the key pair appropriately, is there a way to compromise the security of RSA encryption without recovering the private exponent?

We highlight three concerns, and describe how to deal with each in turn. Among the theoretically interesting results that we do not have space to cover here are those concerned with the protection offered by RSA encryption for various bits of message information. The interested reader can find more information about this topic in the work of Alexi et al. [1].

Though a theoretical concern, the possibility of generating a non-prime is not really a practical issue.

Small Messages

Clearly RSA encryption is not effective on small messages when the public exponent e is small. In particular, when $c = m^e < n$, m can be recovered from c by ordinary root extraction. In fact this attack can be extended somewhat even if there has been some modular reduction by guessing how much reduction has taken place. Thus this attack extended to $c > n$ by trial-and-error might still be faster than exhaustive search for m.

But the precaution is obvious. Either the public exponent e should be large or the messages should always be large. It is this second suggestion that is the most useful since a small public exponent is often preferred. However we have to be careful to ensure that the large message we use is not merely some multiple of a known value such as a large power of two (as would be the case if the message were padded on the right with zeroes). As we will see, this would allow an attacker to mount some sophisticated attacks. So when the public exponent e is small the messages being encrypted should always be large in numerical value and not a multiple of some known value. This can all be achieved by padding the message appropriately prior to encryption.

Either the public exponent e should be large or the messages should always be large.

Chosen Ciphertext Attacks

One of the identities we highlighted at the beginning of this article was:

$$(m * r^e)^d \equiv m^d * r \pmod{n}.$$

An attacker might exploit this fact in the following way. Having intercepted some ciphertext c, the attacker chooses some random number r and computes $r^e \bmod n$. By sending $c * r^e \bmod n$ to the legitimate receiver, $c^d * r \bmod n$ is recovered which will, in all

likelihood, appear to be random. If the attacker were to obtain this decrypted string then the intended ciphertext could be obtained on multiplication by $r^{-1} \bmod n$.

More far-reaching than obtaining the correct decryption of a particular message, is a generalization of this technique by Desmedt and Odlyzko [7]. Since a chosen ciphertext attack on RSA encryption is equivalent in effect to a chosen-text attack on the RSA signature scheme, we shall also see this attack in the next section.

...an attacker should not be able to obtain the raw RSA decryption of an arbitrary value.

The attacker asks the user of RSA to decrypt carefully chosen ciphertexts and obtains a pool of data consisting of the decryptions of small primes and certain other values. Then, when sufficient information has been accumulated, the attacker can decrypt a particular encrypted message by multiplying the ciphertext by a random r^e and factoring the result into small primes and other values in the pool. (The attacker can try another r if unsuccessful.) The decryption of the ciphertext will be the product of the factors' decryptions and $r^{-1} \bmod n$.

While this attack can be more efficient than factoring the modulus, certain precautions can be taken to ensure that it is impractical. However there is already one lesson we can learn, and that is that an attacker should not be able to obtain the raw RSA decryption of an arbitrary value.

Just how practical is a chosen ciphertext attack? Consider an environment where a subscriber to some conditional-access service has access to the decryption equipment but not to the actual keys. In such a case the subscriber might well be free to interrogate the decryption unit at will, with ciphertexts of the subscriber's own choosing. It would clearly be a security flaw if the corresponding decryptions were then directly accessible to the attacker.

Low Exponent Attacks

The use of random padding ... will destroy any known relation between messages.

One class of attacks, perhaps more than any other, has been the cause of confusion about the correct and safe use of RSA. These attacks are due to Hastad, and while the scope and validity of the attacks is not in question, some have taken the existence of these attacks as evidence for avoiding low public exponents in an implementation of RSA.

Hastad [12] showed that if an attacker is able to intercept the encryptions of a single message m generated using several different RSA keys with a common public exponent e, then it might be possible to recover m. As a special case, given l different encryptions $m^e \bmod n_1, \ldots, m^e \bmod n_l$ an attacker can solve for m with the Chinese Remainder Theorem [14].

More generally, given t related encryptions

$$(a_1 m + b_1)^e \bmod n_1, \ldots, (a_t m + b_t)^e \bmod n_t$$

where the a_i's and b_i's are known and $t > e(e+1)/2$, an attacker can solve for m with lattice reduction techniques. Note that this attack is a concern if the messages are related in a known way. The use of sufficient pseudorandom padding prior to encryption will make such an attack impossible to mount in practice, and we will describe one method for doing this in the accompanying box. Messages that are related in a known way should not be encrypted with many RSA keys.

Some very recent work by Franklin and Reiter [10] (see Algorithms Update in this issue of *CryptoBytes*) has highlighted a potential problem when related messages are encrypted under the same RSA key with a low exponent. This work should not be confused with that of Hastad but the problem they ingeniously exploit relies on the fact that the various components required for their attack are once again related in a known way. The use of random padding, as we have already seen, will destroy any known relation between messages. Related messages should not be encrypted with the same RSA key.

Forging Signatures

The problem facing an attacker who is attempting to forge an RSA digital signature is to construct a valid signature for a new message while observing the signatures of other messages that might be known or chosen.

Among the attacks we will consider here are a simple chosen message attack and existential forgery as well as the signature version of the attack on encryption due to Desmedt and Odlyzko. We do not consider issues such as which hash functions are more suitable for use with the RSA signing operation. There are a great many possible designs for hash functions

to choose from [19] and while those based on modular arithmetic might offer certain implementation advantages (since they might rely on much the same operations as are already required for signing), Coppersmith has demonstrated attacks on one such proposal [5]. Extreme care should be taken before considering a hash function based on modular arithmetic for use with RSA signatures.

Chosen and Known Message Attacks

Just as there was a chosen ciphertext attack on encryption, there is the following chosen message attack on raw RSA signatures. As with decryption, once given the signature of $m' = mr^e \bmod n$, where r is random, an attacker can obtain the signature of m since $m^d \equiv (m')^d r^{-1} \pmod n$. Again, the lesson to learn is simple; it should not be possible to obtain a raw RSA signature on an arbitrary value.

Using the mathematical identities at the beginning of the article, it is always possible to compute new message-signature pairs (m, s) of the form

$$m = r^e \prod_{i=1}^{t} m_i^{a_i} \bmod n$$

$$s = r \prod_{i=1}^{t} s_i^{a_i} \bmod n$$

where $(m_1, s_1), \ldots, (m_t, s_t)$ are previous message-signature pairs, and r and $a_1, \ldots, a_t$ are arbitrary. Since the resulting message m is not known in advance, this is an "existential" forgery; the signature exists but the message may or may not be useful. However by selecting messages m_i to be signed, an attacker can derive a signature of a given message m in a chosen message attack. Again, this can be avoided by padding in a way that destroys the algebraic connections between messages.

While a good padding scheme will destroy the algebraic properties which allow these attacks, it is worth pointing out that de Jonge and Chaum [6] have extended such basic attacks to demonstrate vulnerabilities in an early proposal for a simple padding scheme.

Also note that while the algebraic properties we exploit appear to have a negative impact on the use of RSA, we should point out that this "attack" can be used constructively to provide what is called blinding [4] in anonymous payment systems!

Desmedt and Odlyzko's Attack and a Variant

The attack of Desmedt and Odlyzko on encryption applies equally to signatures. Perhaps it is more practical when applied to signature forgery rather than ciphertext decryption, since it might be easier to demand and receive the signature on a variety of messages instead of demanding decryptions for a range of ciphertexts. A variant of this attack can be particularly effective when the messages being signed are small (for instance if they are generated as the output from some message-digest algorithm) unless care is taken.

Essentially, the attack of Desmedt and Odlyzko (for both encryption and signatures) relies on factoring the message into small primes and values near $\sqrt{n}$. An interesting adaptation of this attack (see notes in Section 8.1 of [22]) applies to the case when the numerical input to the signing algorithm is always relatively small. In this adaptation, the attacker factors the particular message into small primes (this may or may not succeed); the signature on the message equals the product of the factors' signatures. The attacker obtains the primes' signatures from appropriate combinations of signatures on other messages whose factors are small. As opposed to when the input is as large as the modulus, values near $\sqrt{n}$ are not needed. The probability of success will therefore depend on the size of the message, not on the size of the RSA modulus.

Messages to be signed should thus be as large as the RSA modulus and, for similar reasons as for encryption, not a multiple of some known value.

RSA Block Formats

Throughout this article we have considered one potential attack after another, in each case highlighting its prevention. Is there a standard way to adopt these safeguards? Is there a way that implementations of RSA can communicate with each other and be assured of a basic level of security?

There are a variety of practical methods for addressing the different recommendations on encryption and signatures. For instance, PKCS #1 [22] (see accompanying box) is a simple, ad-hoc design for protecting RSA encryption and signatures on message digests; it is easy to implement and addresses each of the attacks described above in a heuristic

...it should not be possible to obtain a raw RSA signature on an arbitrary value.

Messages to be signed should be as large as the RSA modulus...

manner. The amount of padding in each block is at least 11 bytes.

ISO/IEC 9796 [13] provides a mathematical design for protecting signatures and also gives message recovery: it is intended for signatures on arbitrary data as well as message digests, and the data or message digest can be recovered from the signature. ISO/IEC 9796 is somewhat more complex than PKCS #1, and about half of each block is padding, but the mathematical motivation is well justified [11].

While many of these attacks would be a serious problem ... we have seen that there is always a practical and efficient countermeasure.

The construction of Bellare and Rogaway [3], also known as Optimal Asymmetric Encryption Padding, provides a cryptographic design for protecting encryption. It involves hash functions and pseudorandom generators and is the most complex of the three approaches. However, it has the benefit that its security can be directly related to the quality of the hash function or pseudorandom generator. The amount of padding is 16 bytes or more, depending on the level of security.

The whole issue of defining block formats is, in itself, an interesting and delicate problem, and deserves more attention than we can give in this article. Instead we refer the reader to the relevant references for additional information.

Conclusions

In this article, we have provided an overview of the major attacks that have resulted from more than 15 years of cryptanalysis. While many of these attacks would be a serious problem if mounted on a raw form of RSA, we have seen that there is always a practical and efficient countermeasure.

References

[1] W. Alexi, B. Chor, O. Goldreich, and C.P. Schnorr. RSA and Rabin functions: certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194-209, April 1988.

[2] D. Atkins, M. Graff, A.K. Lenstra, and P.C. Leyland. The magic words are squeamish ossifrage. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in Cryptology—Asiacrypt '94*, pages 263-277, Springer-Verlag, 1995.

[3] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In A. de Santis, editor, *Advances in Cryptology—Eurocrypt '94*, pages 92-111, Springer-Verlag, 1995.

[4] D. Chaum. Security without identification: transaction systems to make big brother obsolete. *Comm. ACM 28:10*, October, 1985.

[5] D. Coppersmith. *Analysis of ISO/CCITT Document X.509 Annex D*. Internal Memo, IBM T.J. Watson Center, June 11, 1989.

[6] W. de Jonge and D. Chaum. Attacks on some RSA signatures. In H.C. Williams, editor, *Advances in Cryptology—*

PKCS #1 Block Formats

PKCS #1, the RSA signature and encryption standard developed by RSA Laboratories and a number of RSA Data Security's customers in the early 1990's, defines two sets of rules for padding a message block prior to encryption or signature. The padding rules are slightly different in the two cases, because of the different style of attacks that they are intended to protect against.

For encryption, the block format is defined as follows. The length of the entire block is the same as the length of the RSA modulus which is being used. It has the following form (from most significant to least significant byte):

00_x	02_x	PS	00_x	D

The values 00_x and 02_x are byte values in hexadecimal notation, *PS* is a pseudorandom string of nonzero bytes, and *D* is the data to be encrypted. The pseudorandom string must be at least eight bytes long. The byte 00_x ensures that the block is arithmetically less than the RSA modulus. The byte 02_x and the padding makes the input to RSA a large integer, thus preventing short message attacks; the pseudorandomness in *PS* prevents low-exponent attacks. The overall structure of the block prevents the various multiplicative attacks that we observed in the accompanying article.

More specifically, the pseudorandom string increases the difficulty of a variety of attacks—Hastad's, Franklin and Reiter's, and various chosen ciphertext attacks including Desmedt and Odlyzko's—by a factor of at least $(255)^8 \approx 2^{64}$, assuming that the pseudorandom string is discarded by the decryption equipment. To complete any of those attacks, the opponent essentially must guess the bits of the pseudorandom string for at least one, and possibly more messages. (For instance, to complete Franklin and Reiter's attack, the opponent must guess the difference between the encryption blocks for two messages, which involves at least 64 unknown bits.)

The pseudorandom string also thwarts the "verifiable plaintext" attack where an opponent re-encrypts guessed plaintext

Crypto '85, pages 18-27, Springer-Verlag, 1986.

[7] Y.G. Desmedt and A.M. Odlyzko. A chosen text attack on the RSA cryptosystem and some discrete logarithm schemes. In H.C. Williams, editor, *Advances in Cryptology - Crypto '85*, pages 516-522, Springer-Verlag, 1986.

[8] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22: 644-654, 1976.

[9] J.-H. Evertse and E. van Heyst. Which new RSA-signatures can be computed from certain given RSA-signatures? *Journal of Cryptology*, 5(1):41-52, 1992.

[10] M. Franklin and M. Reiter. A linear protocol failure for RSA with exponent three. Presented at the Rump Session of Crypto '95, Santa Barbara, CA.

[11] L.C. Guillou, J.-J. Quisquater, M. Walker, P. Landrock, and C. Shaer. Precautions taken against various potential attacks in ISO/IEC DIS 9796. In I.B. Damgård, editor, *Advances in Cryptology—Eurocrypt '90*, pages 465-473, Springer-Verlag, 1991.

[12] J. Hastad. Solving simultaneous modular equations of low degree. *SIAM Journal on Computing*, 17(2):336-341, April 1988.

[13] ISO/IEC. *International Standard 9796: Information Technology, Security Techniques: Digital Signature Scheme Giving Message Recovery*, 1991.

[14] D.E. Knuth. The Art of Computer Programming, Vol.2: Seminumerical Algorithms. Second edition, Addison-Wesley, Reading, MA., 1981.

[15] H.W. Lenstra Jr. Factoring integers with elliptic curves. *Ann. of Math.*, 126:649-673, 1987.

[16] G.L. Miller. Reimann's hypothesis and tests for primality. *J. Comput. System Sci.*, 13:300-317, 1976.

[17] A.M. Odlyzko. The Future of Integer Factorization. *RSA Laboratories' CryptoBytes*, 1:2, pages 5-12, Summer, 1995.

[18] J. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philos. Soc.*, 76:521-528, 1974.

[19] B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. Ph.D. Thesis, K.U.Leuven. January, 1993.

[20] R. Rivest. Dr. Ron Rivest on the Difficulty of Factoring. *Ciphertext: The RSA Newsletter*, vol. 1, no. 1, fall 1993, and reprinted, in an updated form, in an appendix on pages 361-364 in S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1995.

[21] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120-126, February 1978.

[22] RSA Laboratories. *PKCS #1: RSA Encryption Standard*, Version 1.5, November 1993.

[23] G.J. Simmons and M.J. Norris. Preliminary comments on the MIT public-key cryptosystem. *Cryptologia*, 1(4):406-414, 1977.

[24] M.J. Wiener. Cryptanalysis of short RSA secret exponents. *IEEE Transactions on Information Theory*, 36:553-558, 1990.

[25] H.C. Williams and B. Schmid. Some remarks concerning the MIT public-key cryptosystem. *BIT* 19, pages 525-538, 1979.

and compares the result to the ciphertext. Since the opponent must guess the pseudorandom string in addition, the difficulty increases by at least a factor of 2^{64}.

For protecting RSA signatures, the signature block format is slightly different and has the following form:

00_x	01_x	$ff_x \ldots ff_x$	00_x	D

Again 00_x, 01_x and ff_x are byte values in hexadecimal notation, and *D* is the data to be signed, which must be a message digest. There must be at least eight ff_x bytes.

The byte 00_x again ensures that the block is arithmetically less than the RSA modulus. The byte 01_x and the ff_x padding prevent the variant of Desmedt and Odlyzko's attack on small messages and the overall structure of the block prevents various multiplicative attacks.

Just as for encryption, there is at least a 2^{64} increase in work effort for a variety of attacks. For instance, known and chosen message forgery is 2^{64} harder since an opponent can only obtain "raw" RSA signatures on less than one in every 2^{64} possible messages. The opponent must repeatedly randomize the message to be signed to find one whose block format is correct.

The PKCS #1 block format for signatures is intended only for signatures on the message digests of messages. Some potential attacks arise if the data *D* is arbitrary, since an opponent can select the value of *D* so that the resulting signature block has small prime factors, or perhaps is even a natural power. However, when the data *D* is a message digest, which is effectively a pseudorandom value, these attacks are not a concern; for applications where a message digest is signed with RSA, the format is adequate.

In applications where data is signed directly, a format such as ISO/IEC 9796 designed specifically for signatures with "message recovery" is preferable.

How Do Digital Time-Stamps Support Digital Signatures?

Answered by
Stuart Haber, **Burt Kaliski** and **Scott Stornetta**

Consider two questions that may be asked by a computer user as he or she views a digital document or on-line record. (1) Who is the author of this record — who wrote it, approved it, or consented to it? (2) When was this record created or last modified?

In both cases, the question is one about exactly *this* record—exactly this sequence of bits—whether it was first stored on this computer or was created somewhere else and then copied and saved here. An answer to the first question tells *who & what*: who approved exactly what is in this record? An answer to the second question tells *when & what*: when exactly did the contents of this record first exist?

Both of the above questions have good solutions. A system for answering the first question is called a *digital signature* scheme. Such a system was first proposed in [2] and there is a wide variety of accepted designs for an implementation of this kind of system [4,5].

A system for answering the second question is called a *digital time-stamping scheme*. Such systems were described in [1,3], and an implementation is commercially available from Surety Technologies (*http://www.surety.com/*).

Digital time-stamps increase the longevity of digitally-signed records.

Any system allowing users to answer these questions reliably for all their records must include two different sorts of procedures. First, there must be a *certification* procedure with which (1) the author of a record can "sign" the record, or (2) any user can fix a record in time. The result of this procedure is a small certifying file, a *certificate* if you will, that captures the result of this procedure. Second, there must be a *verification* procedure by which any user can check a record and its accompanying certificate to make sure it correctly answers (1) who and what? or (2) when and what? about the record in question.

The "certificate" returned by the certification procedure of a digital signature system is usually called a *signature*; it is a signature for a particular signer (specifying whom) and for a particular record (specifying what). In order to be able to "sign" documents, a user registers with the system by using special software to compute a pair of numbers called keys, a *public key* and a corresponding *private key*. The private key should only be available to the user to whom it belongs, and is used (by the certification or "signing" procedure) in order to sign documents; it is by employing the user's private key that the signature and the record are tied to that particular user. The public key may be available to many users of the system, and is used by the verification procedure. That is, the verification procedure takes a particular record, a particular user's public key, and a putative signature for that record and that user, and uses this information to check whether the would-be signature was correctly computed using that record and the corresponding private key.

Special computational methods are employed for signing documents and for verifying documents and signatures; when these methods are carefully implemented, they have the remarkable property that the knowledge of a user's public key does not enable an attacker or hacker to figure out the user's corresponding private key. Of course, if, either through carelessness or deliberate intent, someone else—a hacker, for example—gains access to the user's private key, then this person will be able to "forge" the legitimate user's signatures on documents of the hacker's choice. At that point, even the value of legitimately signed records can be called into question.

The "certificate" returned by the certification procedure of a digital time-stamping system is a certificate for a particular record (specifying what) at a particular time (specifying when). The procedure works by mathematically linking the bits of the record to a "summary number" that is widely witnessed by and widely available to members of the public—including, of course, users of the system. The computational methods employed ensure that only the record in question can be linked, according to the "instructions" contained in its time-stamp

Stuart Haber is a research scientist in the Security Research Group at Bellcore, Burt Kaliski is chief scientist at RSA Laboratories, and W. Scott Stornetta is Chairman of Surety Technologies. Surety Technologies was founded by Stornetta and Haber in 1994 as a spin-off from Bellcore, with a mandate to commercialize the digital time-stamping technology developed by Bellcore. The authors can be contacted at stuart@bellcore.com, burt@rsa.com and scotts@surety.com.

certificate, to this widely witnessed summary number; this is how the particular record is tied to a particular moment in time. The verification procedure takes a particular record and a putative time-stamp certificate for that record and a particular time, and uses this information to validate whether that record was indeed certified at the time claimed by checking it against the widely available summary number for that moment.

Two features of a digital time-stamping system are particularly helpful in enhancing the integrity of a digital signature system. First, a time-stamping system cannot be compromised by the disclosure of a key. This is because digital time-stamping systems do not rely on keys, or any other secret information, for that matter. Second, following the technique introduced in [1], digital time-stamp certificates can be *renewed* so as to remain valid indefinitely.

With these features in mind, consider the following situations.

It sometimes happens that the connection between a person and his or her public signature key must be revoked—for example, if the user's secure access to the private key is accidently compromised; or when the key belongs to a job or role in an organization that the person no longer holds. Therefore the person-key connection must have time limits, and the signature verification procedure should check that the record was signed at a time when the signer's public key was indeed in effect. And thus when a user signs a record that may be checked some time later—perhaps after the user's key is no longer in effect—the combination of the record *and* its signature should be certified with a secure digital time-stamping service.

There is another situation in which a user's public key may be revoked. Consider the case of the signer of a particularly important document who later wishes to repudiate his signature. By dishonestly reporting the compromise of his private key, so that all his signatures are called into question, the user is able to disavow the signature he regrets. However, if the document in question was digitally time-stamped together with its signature (and key-revocation reports are time-stamped as well), then the signature cannot easily be disavowed in this way. This is the recommended procedure, therefore, in order to preserve the *non-repudiability* desired of digital signatures for important documents.

The statement that private keys cannot be derived from public keys is an over-simplification of a more complicated situation. In fact, this claim depends on the computational difficulty of certain mathematical problems. As the state of the art advances—both the current state of algorithmic knowledge, as well as the computational speed and memory available in currently available computers—the maintainers of a digital signature system will have to make sure that signers use longer and longer keys. But what is to become of documents that were signed using key lengths that are no longer considered secure? If the signed document is digitally time-stamped, then its integrity can be maintained even after a particular key-length is no longer considered secure.

Time-stamping strengthens non-repudiation of digital signatures.

Of course, digital time-stamp certificates also depend for their security on the difficulty of certain computational tasks concerned with so-called one-way hash functions. (All practical digital-signature systems depend on these functions as well.) Those who maintain a secure digital time-stamping service will have to remain abreast of the state of the art in building and in attacking one-way hash functions. Over time, they will need to upgrade their implementation of these functions, as part of the process of renewal [1]. This will allow time-stamp certificates to remain valid indefinitely.

References

[1] D. Bayer, S. Haber, and W.S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In R.M. Capocelli, A. De Santis, U. Vaccaro, editors, *Sequences II: Methods in Communication, Security, and Computer Science*, pp. 329-334, Springer-Verlag, New York (1993).

[2] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22: 644-654, 1976.

[3] S. Haber and W.S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, Vol. 3, No. 2, pp. 99-111 (1991).

[4] National Institute of Standards and Technology (NIST). FIPS Publication 186: Digital Signature Standard, May 19, 1994.

[5] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120-126, February 1978.

The 1996 RSA Data Security Conference

The 1996 RSA Data Security Conference will be held January 17-19 in the Fairmont Hotel, San Francisco. First held in 1991, this annual conference is expected to attract more than 1,000 participants and provides an excellent opportunity for business people, academic cryptographers and representatives of government to gather and debate the technology and business issues facing the industry.

Spread over three days, there will be a wide range of seminars, tutorials and presentations. Many of the presentations will be focused on the commercial applications of modern cryptographic technology, with an emphasis on Public Key Cryptosystems, but there will also be simultaneous tracks for developers, cryptographers and analysts from which attendees can pick and choose the talks they wish to attend.

There are plans for several open panel discussions. On the final day, the focus of these discussions will be on electronic commerce and doing business on the Internet. Meanwhile Washington will be the topic for discussion on the second day, with issues ranging from CLIPPER, FORTEZZA and key escrow, to export control. Following the keynote address and company announcements on the first morning, a cryptographer's expert panel will provide an excellent opportunity for attendees to direct their questions directly to some of the leading figures in cryptographic research.

Over the years, the RSA Data Security Conference has proved to be a great opportunity for vendors, developers and specialists to meet, and like previous years it is expected to be filled to capacity very soon. More information can be found on RSA's web page (*http://www.rsa.com/*) or by contacting the conference organizer, Layne Kaplan Events, at 415/340-9300.

In this issue:

For subscription information, see page 2 of this newsletter.

RSA LABORATORIES
CRYPTOGRAPHIC RESEARCH AND CONSULTATION
100 MARINE PARKWAY
REDWOOD CITY
CA. 94065-1031
TEL 415/595-7703
FAX 415/595-4126
rsa-labs@rsa.com

PRESORT
FIRST CLASS
U.S. POSTAGE
PAID
MMS, INC