The Mathematics of the RSA Public-Key Cryptosystem

Burt Kaliski
RSA Laboratories

ABOUT THE AUTHOR: Dr Burt Kaliski is a computer scientist whose involvement with the security industry has been through the company that Ronald Rivest, Adi Shamir and Leonard Adleman started in 1982 to commercialize the RSA encryption algorithm that they had invented. At the time, Kaliski had just started his undergraduate degree at MIT. Professor Rivest was the advisor of his bachelor's, master's and doctoral theses, all of which were about cryptography. When Kaliski finished his graduate work, Rivest asked him to consider joining the company, then called RSA Data Security. In 1989, he became employed as the company's first full-time scientist. He is currently chief scientist at RSA Laboratories and vice president of research for RSA Security.

Introduction

Number theory may be one of the "purest" branches of mathematics, but it has turned out to be one of the most useful when it comes to computer security. For instance, number theory helps to protect sensitive data such as credit card numbers when you shop online. This is the result of some remarkable mathematics research from the 1970s that is now being applied worldwide.

Sensitive data exchanged between a user and a Web site needs to be encrypted to prevent it from being disclosed to or modified by unauthorized parties. The encryption must be done in such a way that decryption is only possible with knowledge of a secret *decryption key*. The decryption key should only be known by authorized parties.

In traditional cryptography, such as was available prior to the 1970s, the encryption and decryption operations are performed with the same key. This means that the party encrypting the data and the party decrypting it need to share the same decryption key. Establishing a shared key between the parties is an interesting challenge. If two parties already share a secret key, they could easily distribute new keys to each other by encrypting them with prior keys. But if they don't already share a secret key, how do they establish the first one?

This challenge is relevant to the protection of sensitive data on the Web and many other applications like it. Your computer doesn't initially share any secret keys with Web sites. How then do you encrypt data you are sending to the site? You might eventually set up a password, and the password could then be used to derive an encryption key. But how do you protect the password from interception when you're first setting it up?

This line of thinking – in pre-Web terminology – prompted two Stanford University researchers, Whitfield Diffie and Martin Hellman, to write a landmark paper, “New Directions in Cryptography,” in 1976. The paper suggested that perhaps encryption and decryption could be done with a pair of *different* keys rather than with the same key. The decryption key would still have to be kept secret, but the encryption key could be made public without compromising the security of the decryption key. This concept was called *public-key cryptography* because of the fact that the encryption key could be made known to anyone.

Diffie and Hellman’s approach immediately answered the setup problems for protecting online data. To enable computers to encrypt data for a site, the site simply needs to publish its encryption key, for instance in a directory. Every computer can use that encryption key to protect data sent to the site. But only the site has the corresponding decryption key, so only it can decrypt the data.

Diffie and Hellman also introduced the concept of *digital signatures*. Parties who share a secret key can easily verify that data they exchange has not been modified by performing an authentication operation using the key. But what if they don’t share a key?

For this problem, Diffie and Hellman suggested another application of public-key cryptography, but with the order of encryption and decryption reversed. To protect the data it sends from modification, a site would first “decrypt” the data with its private decryption key. The computer receiving the result would “encrypt” it with the corresponding encryption key, and thereby recover and verify the data.

The result is called a digital signature because it has similar properties to handwritten signatures: Any computer can verify the signature, but only the originating site can generate it. In fact the assurances are stronger than for ordinary signatures, because the signature is dependent on the data itself, unlike ordinary signatures which can potentially be cut-and-pasted among different documents and yet still appear to be valid.

Digital signatures help with the next question in public-key cryptography: If a site publishes its encryption key in a directory, how do I know that the data in the directory is authentic? Answer: The directory digitally signs the data with its private key. All the computer needs to have in advance is the directory’s public key for verifying the directory’s signature. (This concept is implemented today in *digital certificates*.) The computer can then verify the encryption key for any site.

Diffie and Hellman didn’t identify a method with the full public-key encryption/decryption properties that would also enable digital signatures. However, they did introduce a specific method based on number theory for establishing secret keys by parties who don’t previously share a secret. The method, called *Diffie-Hellman key agreement*, is still in use today. The security of the method is related to a longstanding problem in number theory, *discrete logarithms*. The full public-key method would come a year later as an application of another famous problem, *integer factorization*.

Prime Generation and Integer Factorization

Two basic facts and one conjecture in number theory prepare the way for today's RSA public-key cryptosystem.

FACT 1. **Prime generation is easy:** It's easy to find a random prime number of a given size.

This is a result of two other points: Prime numbers of any size are very common, and it's easy to test whether a number is a prime – even a large prime.

To generate a random prime, one can simply generate random numbers of a given size and test them for primality until a prime is found. According to the Prime Number Theorem, the expected number of candidates to test will be on the order of ln x (the natural logarithm of x) where x is a typical number of the intended size.

It hasn't always been easy to test whether a number is a prime. In fact, it might seem that testing for primality would require one to determine all the factors of the number to see if there are others beside the number itself and 1. Faster methods for primality testing were discovered in the 1970s that test for certain properties held by prime numbers but not by composites, rather than finding the factors. Without these results, much of public-key cryptography today would not be practical because of the dependence on efficient methods of generating primes.

In the following, let p and q be two large, randomly generated primes. "Large" in the cryptographic context typically means 512 bits (155 decimal digits) or more.

FACT 2. **Multiplication is easy:** Given p and q, it's easy to find their product, $n = pq$.

There are many efficient ways to multiply two large numbers, starting with the "grade-school" method that multiplies one number by the other digit-by-digit, and sums the tableau of intermediate results.

CONJECTURE 3. **Factoring is hard:** Given such an n, it appears to be quite hard to recover the prime factors p and q.

Despite hundreds of years of study of the problem, finding the factors of a large number still takes a long time in general. The fastest current methods are much faster than the simple approach of trying all possible factors one at a time. (Such a method would take on the order of $\sqrt{n}$ steps.) However, they are still expensive. For instance, it has been estimated recently that recovering the prime factors of a 1024-bit number would take a year on a machine costing US $10 million. A 2048-bit number would require several *billion* times more work.

These estimates are much less than would have been expected in the 1970s when the problem was first proposed in cryptography. The recommended sizes have accordingly increased over the years, due to the discovery of faster factoring methods as well as steady advances in computing power.

No one knows whether still faster methods might be discovered in the coming years. On the other hand, no one has proved that they can't be. Both aspects remain important research areas in mathematics.

Modular Exponentiation and Roots

Given this background, n will hereafter denote the product of two large, randomly generated primes. Let m and c be integers between 0 and n-1, and let e be an odd integer between 3 and n-1 that is relatively prime to p-1 and q-1.

The encryption and decryption operations in the RSA public-key cryptosystem are based on two more facts and one more conjecture:

FACT 4. **Modular exponentiation is easy:** Given n, m, and e, it's easy to compute $c = m^e \bmod n$.

The value $m^e \bmod n$ is formally the result of multiplying e copies of m, dividing by n, and keeping the remainder. This may seem to be an expensive computation, involving e-1 multiplications by m with increasingly large intermediate results, followed by a division by n. However, two optimizations make the operation easy:

1. Multiplying by an appropriate sequence of previous intermediate values, rather than only by m, can reduce the number of multiplications to no more than twice the *size* of e in binary.
2. Dividing and taking the remainder after each multiplication keeps the intermediate results the same size as n.

FACT 5. **Modular root extraction – the reverse of modular exponentiation – is easy given the prime factors:** Given n, e, c, and the prime factors p and q, it's easy to recover the value m such that $c = m^e \bmod n$.

The value m can be recovered from c by a modular exponentiation operation with another odd integer d between 3 and n-1. In particular, for this d, the following holds for all m:

$$m = (m^e)^d \bmod n \; .$$

This integer d is easy to compute given e, p, and q; see below for details.

CONJECTURE 6. **Modular root extraction is otherwise hard:** Given only n, e, and c, but not the prime factors, it appears to be quite hard to recover the value m.

The fastest general method currently available for computing modular roots under the conditions on n and e above is to factor n and apply FACT 5 to determine d. Actually, any method that determines the value d can be turned into a method for factoring n. It's possible that there may be methods that compute modular roots without factoring n or determining d. But so far no general methods have been found for doing so that are faster than factoring n.

[Note: In some cases, it is easy to compute modular roots without knowledge of the prime factors. For instance, if m is known to be very small, such that $c = m^e < n$, then m can be recovered from c by taking e^{th} roots over the integers, which is easy. However, these cases are very rare in practice for typical constructions of the plaintext m. A general root-extraction method must be able to compute modular roots for many plaintexts, not just a few.]

The RSA Cryptosystem

The various observations just stated form the basis for the *RSA public-key cryptosystem*, which was invented at MIT in 1977 by Ronald Rivest, Adi Shamir and Leonard Adleman.

The public key in this cryptosystem consists of the value n, which is called the *modulus*, and the value e, which is called the *public exponent*. The private key consists of the modulus n and the value d, which is called the *private exponent*.

An RSA public-key / private-key pair can be generated by the following steps:

1. Generate a pair of large, random primes p and q.
2. Compute the modulus n as $n = pq$.
3. Select an odd public exponent e between 3 and n-1 that is relatively prime to p-1 and q-1.
4. Compute the private exponent d from e, p and q. (See below.)
5. Output (n, e) as the public key and (n, d) as the private key.

The encryption operation in the RSA cryptosystem is exponentiation to the e^{th} power modulo n:

$$c = \textsc{Encrypt}\ (m) = m^e \bmod n \ \ .$$

The input m is the *message*; the output c is the resulting *ciphertext*. In practice, the message m is typically some kind of appropriately formatted key to be shared. The actual message is encrypted with the shared key using a traditional encryption algorithm. This construction makes it possible to encrypt a message of any length with only one exponentiation.

The decryption operation is exponentiation to the d^{th} power modulo n:

$$m = \textsc{Decrypt}\ (c) = c^d \bmod n \ \ .$$

The relationship between the exponents e and d ensures that encryption and decryption are inverses, so that the decryption operation recovers the original message m. Without the private key (n, d) (or equivalently the prime factors p and q), it's difficult (by Conjecture 6) to recover m from c. Consequently, n and e can be made public without compromising security, which is the basic requirement for a public-key cryptosystem.

The fact that the encryption and decryption operations are inverses and operate on the same set of inputs also means that the operations can be employed in reverse order to obtain a digital signature scheme following Diffie and Hellman's model. A message can be digitally signed by applying the decryption operation to it, i.e., by exponentiating it to the d^{th} power:

$$s = \text{Sign}\,(m) = m^d \bmod n \ \ .$$

The digital signature can then be verified by applying the encryption operation to it and comparing the result with and/or recovering the message:

$$m = \text{Verify}\,(s) = s^e \bmod n \ \ .$$

In practice, the plaintext m is generally some function of the message, for instance a formatted one-way hash of the message. This makes it possible to sign a message of any length with only one exponentiation.

Figure 1 gives a small example showing the encryption of values m from 0 to 9 as well as decryptions of the resulting ciphertexts. The exponentiation is optimized as suggested above. To compute $m^3 \bmod n$, one first computes $m^2 \bmod n$ with one modular squaring, then $m^3 \bmod n$ with a modular multiplication by m. The decryption is done similarly: One first computes $c^2 \bmod n$, then $c^3 \bmod n$, $c^6 \bmod n$, and $c^7 \bmod n$ by alternating modular squaring and modular multiplication.

Key Pair *Public key:* $n = 55, e = 3$ *Private key:* $n = 55, d = 7$			***Key Pair Generation*** *Primes:* $p = 5, q = 11$ *Modulus:* $n = pq = 55$ *Public exponent:* $e = 3$ *Private exponent:* $d = 3^{-1} \bmod 20 = 7$			
Message	***Encryption*** $c = m^3 \bmod n$		***Decryption*** $m = c^7 \bmod n$			
m	$m^2 \bmod n$	$m^3 \bmod n$	$c^2 \bmod n$	$c^3 \bmod n$	$c^6 \bmod n$	$c^7 \bmod n$
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	4	8	9	17	14	2
3	9	27	14	48	49	3
4	16	9	26	14	31	4
5	25	15	5	20	15	5
6	36	51	16	46	26	6
7	49	13	4	52	9	7

8	9	17	14	18	49	8
9	26	14	31	49	36	9

Figure 1: Small example of the RSA public-key cryptosystem.

From Theory to Practice

It has been a long road from Diffie and Hellman's discovery of public-key cryptography in 1976 and the invention of the RSA public-key cryptosystem in 1977, to the widespread deployment of public-key cryptography we see today.

Public-key cryptography finds its strongest application when parties who have no prior relationship (and therefore no opportunity to establish shared secret keys) want to exchange sensitive data with each other. Throughout the 1980s, however, most of the applications that needed to protect data had centralized control. Banking networks and pay-TV systems are typical examples where secret keys could generally be pre-established by a central authority. Applications that didn't have centralized control – like e-mail – were meanwhile growing without much attention to security. Equally important was the fact that the mathematical operations in public-key cryptography required considerable computational resources relative to computer performance at the time. As a result, public-key cryptography was a slow sell through that first full decade.

(This was probably a good thing, because cryptographers were still learning what it would mean for a public-key cryptosystem to be "secure," and many of the initial proposals for applying the RSA algorithm would later turn out not to be.)

With the advent of the World Wide Web in the 1990s, however, the situation changed. Computer performance had by then advanced to the point that the time for the encryption and decryption operations was no longer an issue. Meanwhile, the "killer application" of online purchasing had exactly the characteristics that required public-key cryptography. The Web inherently didn't have central control for security: Any merchant could go online without any prior security relationships with anyone else. And there was clearly a need to protect sensitive data: Many consumers would not shop online if there were a risk that credit card numbers and order information might be intercepted by an eavesdropper. Public-key cryptography caught on rapidly as a result.

I've had the privilege to observe the emergence of public-key cryptography in the industry as a scientist at RSA Laboratories, the research center of RSA Security. RSA Laboratories was founded in 1991 to support the start-up company RSA Data Security with research and industry standards development in cryptography. RSA Data Security was acquired by Security Dynamics in 1996, and the parent company changed its name to RSA Security in 1999. The team of eight researchers I now work with at RSA Laboratories supports the combined RSA Security with research and standards activities in user authentication as well.

Our research on the RSA algorithm brings a significant amount of mathematics to bear. For instance, we have needed to understand how best to use the algorithm to establish keys and sign messages. This requires a careful analysis of various methods for relating of the value *m* to the actual message or key, some of which are much better for security than others. We have needed to understand the impact of various proposed improvements to integer factorization methods, especially in terms of recommended key sizes. This requires assessment of the effectiveness of those methods.

Because of the widespread deployment of the RSA algorithm, many other researchers are looking at these same problems today. We benefit significantly from their work as we look to improve our own products and provide guidance to our customers. Our work in industry standards aims to promote the adoption of the best practices we have learned.

The Work Has Just Begun

Simple concepts in mathematics – prime numbers, integer factorization, modular exponentiation – have had a dramatic impact on computer security, particularly for online commerce. The theory is working well in practice through algorithms like Diffie-Hellman key agreement, the RSA public-key cryptosystem and more recently *elliptic curve cryptography*.

In cryptography, "it's not broken" is no reason to avoid trying to fix it. Mathematicians still don't know whether or not there are faster methods for integer factorization than the ones currently available. Research is needed to try to find faster methods, as well to try to prove that there aren't any. A related research problem is to confirm whether modular root-extraction is or isn't as hard as integer factorization.

Interestingly, much faster methods for integer factorization already exist in theory, but they run on computers that haven't yet been built. In particular, if one could build a full-scale *quantum computer*, it will be possible to break a large number into its factors essentially as easily as it is to put the number together by multiplication. (Such a computer would also break the Diffie-Hellman and elliptic curve algorithms.)

In case one or more of the current public-key cryptosystems is broken in the future, it would be helpful to have alternatives to choose from. This is another important area for research. What other hard problems in mathematics are there from which a public-key cryptosystem and digital signature scheme might be derived?

Mathematics has many more applications in computer security than just public-key cryptography, of course. The design and analysis of more traditional encryption algorithms and one-way function also has a strong mathematical component, although perhaps not one so elegant as for public-key cryptography. Intriguing mathematical constructions have also led to new types of cryptography – like *identity-based encryption*, a form of public-key cryptography where one's name or e-mail address itself becomes the public key, avoiding the need for a directory. More groundbreaking applications are likely to emerge over time as knowledgeable people continue to search what's concealed within mathematics. Who knows what other useful things we might find by exploring an otherwise obscure formula?

Appendix: Computing the Private Exponent

Let n be the product of two distinct prime numbers p and q, and let e be the public exponent as defined above. Let L = LCM (p-1, q-1) denote the least common multiple of p-1 and q-1. The private exponent d for the RSA cryptosystem is any integer solution to the congruence

$$de \equiv 1 \bmod L.$$

The value d is the inverse of e modulo L. The requirement that e be relatively prime to p-1 and q-1 ensures that an inverse exists. Modular inverses are easy to find with the Extended Euclidean Algorithm or similar methods.

The RSA cryptosystem works because exponentiation to the d^{th} power modulo n is the inverse of exponentiation to the e^{th} power when the exponents d and e are inverses modulo L. That is, for all m between 0 and n-1,

$$m \equiv (m^e)^d \bmod n \quad .$$

The proof of this fact is left as an exercise to the reader. *Hint:* Show that the result holds modulo p and q separately, i.e., that for all m,

$$m \equiv (m^e)^d \bmod p$$

and

$$m \equiv (m^e)^d \bmod q \quad .$$

The result will then follow via the Chinese Remainder Theorem.