



**WIKIPEDIA**  
The Free Encyclopedia

[Main page](#)  
[Contents](#)  
[Featured content](#)  
[Current events](#)  
[Random article](#)  
[Donate to Wikipedia](#)  
[Wikipedia store](#)

**Interaction**  
[Help](#)  
[About Wikipedia](#)  
[Community portal](#)  
[Recent changes](#)  
[Contact page](#)

**Tools**  
[What links here](#)  
[Related changes](#)  
[Upload file](#)  
[Special pages](#)  
[Permanent link](#)  
[Page information](#)  
[Wikidata item](#)  
[Cite this page](#)

**Print/export**  
[Create a book](#)  
[Download as PDF](#)  
[Printable version](#)

**Languages**  
العربية  
Беларуская (тарашкевіца)  
Български  
Bosanski  
Català  
Čeština  
Dansk  
Deutsch  
Eesti  
Ελληνικά  
Español  
Esperanto  
Euskara  
فارسی  
Français  
Galego  
한국어  
Հայերեն  
Hrvatski  
Bahasa Indonesia  
Íslenska  
Italiano  
עברית  
Қазақша  
Latviešu  
Lietuvių  
Magyar  
Bahasa Melayu

# RSA (cryptosystem)

From Wikipedia, the free encyclopedia  
(Redirected from [RSA \(algorithm\)](#))

*For other uses, see [RSA \(disambiguation\)](#).*

**RSA** is one of the first practical [public-key cryptosystems](#) and is widely used for secure data transmission. In such a [cryptosystem](#), the [encryption key](#) is public and differs from the [decryption key](#) which is kept secret. In RSA, this asymmetry is based on the practical difficulty of [factoring](#) the product of two large [prime numbers](#), the [factoring problem](#). RSA is made of the initial letters of the surnames of [Ron Rivest](#), [Adi Shamir](#), and [Leonard Adleman](#), who first publicly described the algorithm in 1977. [Clifford Cocks](#), an English mathematician, had developed an equivalent system in 1973, but it was not [declassified](#) until 1997.<sup>[1]</sup>

A user of RSA creates and then publishes a public key based on two large [prime numbers](#), along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime numbers can feasibly decode the message.<sup>[2]</sup> Breaking RSA [encryption](#) is known as the [RSA problem](#); whether it is as hard as the factoring problem remains an open question.

## RSA

### General

**Designers** [Ron Rivest](#), [Adi Shamir](#), and [Leonard Adleman](#)  
**First published** 1977  
**Certification** [PKCS#1](#), [ANSI X9.31](#), [IEEE 1363](#)

### Cipher detail

**Key sizes** 1,024 to 4,096 bit typical  
**Rounds** 1

### Best public cryptanalysis

[General number field sieve](#) for classical computers  
[Shor's algorithm](#) for quantum computers  
A 768-bit key has been broken

## Contents [hide]

- 1 History
- 2 Patent
- 3 Operation
  - 3.1 Key generation
  - 3.2 Encryption
  - 3.3 Decryption
  - 3.4 A worked example
  - 3.5 Signing messages
- 4 Proofs of correctness
  - 4.1 Proof using Fermat's little theorem
  - 4.2 Proof using Euler's theorem
- 5 Padding
  - 5.1 Attacks against plain RSA
  - 5.2 Padding schemes
- 6 Security and practical considerations
  - 6.1 Using the Chinese remainder algorithm
  - 6.2 Integer factorization and RSA problem
  - 6.3 Faulty key generation
  - 6.4 Importance of strong random number generation
  - 6.5 Timing attacks
  - 6.6 Adaptive chosen ciphertext attacks
  - 6.7 Side-channel analysis attacks
- 7 See also
- 8 Notes
- 9 References
- 10 Further reading
- 11 External links

## History [edit]

The idea of an asymmetric public-private key cryptosystem is attributed to [Diffie](#) and [Hellman](#), who published the

Μονηρον  
Nederlands  
日本語  
Norsk bokmål  
Polski  
Português  
Română  
Русский  
Simple English  
Slovenščina  
Српски / srpski  
Suomi  
Svenska  
ไทย  
Türkçe  
Українська  
Tiếng Việt  
中文


 Edit links

concept in 1976. The same two also introduced digital signatures and attempted to apply number theory. Their formulation used a shared secret key created from exponentiation of some number, modulo a prime number. However, they left open the problem of realizing a one-way function, possibly because the difficulty of factoring was not well studied at the time.<sup>[3]</sup>

Ron Rivest, Adi Shamir, and Leonard Adleman at MIT made several attempts over the course of a year to create a one-way function that is hard to inverse. Rivest and Shamir, as computer scientists, proposed many potential functions while Adleman, as a mathematician, was responsible for finding their weaknesses. They tried many approaches including "knapsack-based" and "permutation polynomials". At a time they thought it was impossible for what they wanted to achieve due to contradictory requirements.<sup>[4]</sup> In April 1977, they spent Passover at the house of a student and drank a good deal of Manischewitz wine before returning to their home at around midnight.<sup>[5]</sup> Rivest, unable to sleep, lay on the couch with a math textbook and started thinking about their one-way function. He spent the rest of the night formalizing his idea and had much of the paper ready by daybreak. The algorithm is now known as RSA – the initials of their surnames in same order as their paper.<sup>[6]</sup>

Clifford Cocks, an English mathematician working for the UK intelligence agency GCHQ, described an equivalent system in an internal document in 1973. However, given the relatively expensive computers needed to implement it at the time, it was mostly considered a curiosity and, as far as is publicly known, was never deployed. His discovery, however, was not revealed until 1997 due to its top-secret classification.

## Patent <sup>[edit]</sup>

MIT was granted U.S. Patent 4,405,829  for a "Cryptographic communications system and method" that used the algorithm, on September 20, 1983. Though the patent was going to expire on September 21, 2000 (the term of patent was 17 years at the time), the algorithm was released to the public domain by RSA Security on September 6, 2000, two weeks earlier.<sup>[7]</sup> Since a paper describing the algorithm had been published in August 1977,<sup>[6]</sup> prior to the December 1977 filing date of the patent application, regulations in much of the rest of the world precluded patents elsewhere and only the US patent was granted. Had Cocks' work been publicly known, a patent in the US would not have been possible either.

From the DWPI's abstract of the patent,

The system includes a communications channel coupled to at least one terminal having an encoding device and to at least one terminal having a decoding device. A message-to-be-transferred is enciphered to ciphertext at the encoding terminal by encoding the message as a number M in a predetermined set. That number is then raised to a first predetermined power (associated with the intended receiver) and finally computed. The remainder or residue, C, is... computed when the exponentiated number is divided by the product of two predetermined prime numbers (associated with the intended receiver).

## Operation <sup>[edit]</sup>


The RSA algorithm involves three steps: key generation, encryption and decryption.

### Key generation <sup>[edit]</sup>

RSA involves a *public key* and a *private key*. The public key can be known by everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers *p* and *q*.
  - For security purposes, the integers *p* and *q* should be chosen at random, and should be of similar bit-length. Prime integers can be efficiently found using a primality test.
2. Compute  $n = pq$ .
  - *n* is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute  $\phi(n) = \phi(p)\phi(q) = (p - 1)(q - 1) = n - (p + q - 1)$ , where  $\phi$  is Euler's totient function. This value is kept private.



Adi Shamir, one of the authors of RSA: Rivest, Shamir and Adleman 

- Choose an integer  $e$  such that  $1 < e < \phi(n)$  and  $\gcd(e, \phi(n)) = 1$ ; i.e.,  $e$  and  $\phi(n)$  are **coprime**.
  - $e$  is released as the public key exponent.
  - $e$  having a short **bit-length** and small **Hamming weight** results in more efficient encryption – most commonly  $2^{16} + 1 = 65,537$ . However, much smaller values of  $e$  (such as 3) have been shown to be less secure in some settings.<sup>[8]</sup>
- Determine  $d$  as  $d \equiv e^{-1} \pmod{\phi(n)}$ ; i.e.,  $d$  is the **modular multiplicative inverse** of  $e$  (modulo  $\phi(n)$ ).
  - This is more clearly stated as: solve for  $d$  given  $d \cdot e \equiv 1 \pmod{\phi(n)}$
  - This is often computed using the **extended Euclidean algorithm**. Using the pseudocode in the **Modular integers** section, inputs  $a$  and  $n$  correspond to  $e$  and  $\phi(n)$ , respectively.
  - $d$  is kept as the private key exponent.

The *public key* consists of the modulus  $n$  and the public (or encryption) exponent  $e$ . The *private key* consists of the modulus  $n$  and the private (or decryption) exponent  $d$ , which must be kept secret.  $p$ ,  $q$ , and  $\phi(n)$  must also be kept secret because they can be used to calculate  $d$ .

- An alternative, used by **PKCS#1**, is to choose  $d$  matching  $de \equiv 1 \pmod{\lambda}$  with  $\lambda = \text{lcm}(p-1, q-1)$ , where  $\text{lcm}$  is the **least common multiple**. Using  $\lambda$  instead of  $\phi(n)$  allows more choices for  $d$ .  $\lambda$  can also be defined using the **Carmichael function**,  $\lambda(n)$ .

Since any common factors of  $(p-1)$  and  $(q-1)$  are present in the factorisation of  $p \cdot q - 1$ ,<sup>[9]</sup> it is recommended that  $(p-1)$  and  $(q-1)$  have only very small common factors, if any besides the necessary 2.<sup>[10] [11]</sup>

## Encryption <sup>[edit]</sup>

Alice transmits her public key  $(n, e)$  to Bob and keeps the private key  $d$  secret. Bob then wishes to send message  $M$  to Alice.

He first turns  $M$  into an integer  $m$ , such that  $0 \leq m < n$  and  $\gcd(m, n) = 1$  by using an agreed-upon reversible protocol known as a **padding scheme**. He then computes the ciphertext  $c$  corresponding to

$$c \equiv m^e \pmod{n}$$

This can be done efficiently, even for 500-bit numbers, using **Modular exponentiation**. Bob then transmits  $c$  to Alice.

Note that at least nine values of  $m$  will yield a ciphertext  $c$  equal to  $m$ ,<sup>[note 1]</sup>

## Decryption <sup>[edit]</sup>

Alice can recover  $m$  from  $c$  by using her private key exponent  $d$  via computing

$$m \equiv c^d \pmod{n}$$

Given  $m$ , she can recover the original message  $M$  by reversing the padding scheme.

(In practice, there are more efficient methods of calculating  $c^d$  using the precomputed values **below**.)

## A worked example <sup>[edit]</sup>

Here is an example of RSA encryption and decryption. The parameters used here are artificially small, but one can also **use OpenSSL to generate and examine a real keypair**.

- Choose two distinct prime numbers, such as

$$p = 61 \text{ and } q = 53$$

- Compute  $n = pq$  giving

$$n = 61 \times 53 = 3233$$

- Compute the **totient** of the product as  $\phi(n) = (p-1)(q-1)$  giving

$$\phi(3233) = (61-1)(53-1) = 3120$$

- Choose any number  $1 < e < 3120$  that is **coprime** to 3120. Choosing a prime number for  $e$  leaves us only to check that  $e$  is not a divisor of 3120.

$$\text{Let } e = 17$$

- Compute  $d$ , the **modular multiplicative inverse** of  $e \pmod{\phi(n)}$  yielding,

$$d = 2753$$

Worked example for the modular multiplicative inverse:

$$e \times d \pmod{\phi(n)} = 1$$

$$17 \times 2753 \pmod{3120} = 1$$

The **public key** is ( $n = 3233$ ,  $e = 17$ ). For a padded [plaintext](#) message  $m$ , the encryption function is

$$c(m) = m^{17} \bmod 3233$$

The **private key** is ( $d = 2753$ ). For an encrypted [ciphertext](#)  $c$ , the decryption function is

$$m(c) = c^{2753} \bmod 3233$$

For instance, in order to encrypt  $m = 65$ , we calculate

$$c = 65^{17} \bmod 3233 = 2790$$

To decrypt  $c = 2790$ , we calculate

$$m = 2790^{2753} \bmod 3233 = 65$$

Both of these calculations can be computed efficiently using the [square-and-multiply algorithm](#) for [modular exponentiation](#). In real-life situations the primes selected would be much larger; in our example it would be trivial to factor  $n$ , 3233 (obtained from the freely available public key) back to the primes  $p$  and  $q$ . Given  $e$ , also from the public key, we could then compute  $d$  and so acquire the private key.

Practical implementations use the [Chinese remainder theorem](#) to speed up the calculation using modulus of factors ( $\bmod pq$  using  $\bmod p$  and  $\bmod q$ ).

The values  $d_p$ ,  $d_q$  and  $q_{\text{inv}}$ , which are part of the private key are computed as follows:

$$d_p = d \bmod (p - 1) = 2753 \bmod (61 - 1) = 53$$

$$d_q = d \bmod (q - 1) = 2753 \bmod (53 - 1) = 49$$

$$q_{\text{inv}} = q^{-1} \bmod p = 53^{-1} \bmod 61 = 38$$

$$\Rightarrow (q_{\text{inv}} \times q) \bmod p = 38 \times 53 \bmod 61 = 1$$

Here is how  $d_p$ ,  $d_q$  and  $q_{\text{inv}}$  are used for efficient decryption. (Encryption is efficient by choice of public exponent  $e$ )

$$m_1 = c^{d_p} \bmod p = 2790^{53} \bmod 61 = 4$$

$$m_2 = c^{d_q} \bmod q = 2790^{49} \bmod 53 = 12$$

$$h = (q_{\text{inv}} \times (m_1 - m_2)) \bmod p = (38 \times -8) \bmod 61 = 1$$

$$m = m_2 + h \times q = 12 + 1 \times 53 = 65$$

### Signing messages [\[edit\]](#)

Suppose [Alice](#) uses [Bob's](#) public key to send him an encrypted message. In the message, she can claim to be Alice but Bob has no way of verifying that the message was actually from Alice since anyone can use Bob's public key to send him encrypted messages. In order to verify the origin of a message, RSA can also be used to [sign](#) a message.

Suppose Alice wishes to send a signed message to Bob. She can use her own private key to do so. She produces a [hash value](#) of the message, raises it to the power of  $d$  (modulo  $n$ ) (as she does when decrypting a message), and attaches it as a "signature" to the message. When Bob receives the signed message, he uses the same hash algorithm in conjunction with Alice's public key. He raises the signature to the power of  $e$  (modulo  $n$ ) (as he does when encrypting a message), and compares the resulting hash value with the message's actual hash value. If the two agree, he knows that the author of the message was in possession of Alice's private key, and that the message has not been tampered with since.

### Proofs of correctness [\[edit\]](#)

#### Proof using Fermat's little theorem [\[edit\]](#)

The proof of the correctness of RSA is based on [Fermat's little theorem](#). This theorem states that if  $p$  is prime and  $p$  does not divide an integer  $a$  then

$$a^{p-1} \equiv 1 \pmod{p}$$

We want to show that  $m^{ed} \equiv m \pmod{pq}$  for every integer  $m$  when  $p$  and  $q$  are distinct prime numbers and  $e$  and  $d$  are positive integers satisfying

$$ed \equiv 1 \pmod{\phi(pq)}.$$

Since  $\phi(pq) = (p-1)(q-1)$ , we can write

$$ed - 1 = h(p - 1)(q - 1)$$

for some nonnegative integer  $h$ .

To check whether two numbers, like  $m^{ed}$  and  $m$ , are congruent mod  $pq$  it suffices (and in fact is equivalent) to check they are congruent mod  $p$  and mod  $q$  separately. (This is part of the [Chinese remainder theorem](#), although it is not the significant part of that theorem.) To show  $m^{ed} \equiv m \pmod{p}$ , we consider two cases:  $m \equiv 0 \pmod{p}$  and  $m \not\equiv 0 \pmod{p}$ .

In the first case  $m^{ed}$  is a multiple of  $p$ , so  $m^{ed} \equiv 0 \equiv m \pmod{p}$ . In the second case

$$m^{ed} = m^{(ed-1)}m = m^{h(p-1)(q-1)}m = \left(m^{p-1}\right)^{h(q-1)}m \equiv 1^{h(q-1)}m \equiv m \pmod{p}$$

where we used [Fermat's little theorem](#) to replace  $m^{p-1} \pmod{p}$  with 1.

The verification that  $m^{ed} \equiv m \pmod{q}$  proceeds in a similar way, treating separately the cases  $m \equiv 0 \pmod{q}$  and  $m \not\equiv 0 \pmod{q}$ , using Fermat's little theorem for modulus  $q$  in the second case.

This completes the proof that, for any integer  $m$ , and integers  $e, d$  such that  $ed \equiv 1 \pmod{\phi(pq)}$ ,

$$(m^e)^d \equiv m \pmod{pq}$$

### Proof using Euler's theorem [\[edit\]](#)

Although the original paper of Rivest, Shamir, and Adleman used Fermat's little theorem to explain why RSA works, it is common to find proofs that rely instead on [Euler's theorem](#).

We want to show that  $m^{ed} \equiv m \pmod{n}$ , where  $n = pq$  is a product of two different prime numbers and  $e$  and  $d$  are positive integers satisfying  $ed \equiv 1 \pmod{\phi(n)}$ . Since  $e$  and  $d$  are positive, we can write  $ed = 1 + h\phi(n)$  for some non-negative integer  $h$ . Assuming that  $m$  is relatively prime to  $n$ , we have

$$m^{ed} = m^{1+h\phi(n)} = m \left(m^{\phi(n)}\right)^h \equiv m(1)^h \equiv m \pmod{n}$$

where the second-last congruence follows from [Euler's theorem](#).

When  $m$  is not relatively prime to  $n$ , the argument just given is invalid. This is highly improbable (only a proportion of  $1/p + 1/q - 1/(pq)$  numbers have this property), but even in this case the desired congruence is still true. Either  $m \equiv 0 \pmod{p}$  or  $m \equiv 0 \pmod{q}$ , and these cases can be treated using the previous proof.

## Padding [\[edit\]](#)

### Attacks against plain RSA [\[edit\]](#)

There are a number of attacks against plain RSA as described below.

- When encrypting with low encryption exponents (e.g.,  $e = 3$ ) and small values of the  $m$ , (i.e.,  $m < n^{1/e}$ ) the result of  $m^e$  is strictly less than the modulus  $n$ . In this case, ciphertexts can be easily decrypted by taking the  $e$ th root of the ciphertext over the integers.
- If the same clear text message is sent to  $e$  or more recipients in an encrypted way, and the receivers share the same exponent  $e$ , but different  $p, q$ , and therefore  $n$ , then it is easy to decrypt the original clear text message via the [Chinese remainder theorem](#). Johan Håstad noticed that this attack is possible even if the cleartexts are not equal, but the attacker knows a linear relation between them.<sup>[12]</sup> This attack was later improved by Don Coppersmith.<sup>[13]</sup>  
See also: [Coppersmith's Attack](#)
- Because RSA encryption is a [deterministic encryption algorithm](#) (i.e., has no random component) an attacker can successfully launch a [chosen plaintext attack](#) against the cryptosystem, by encrypting likely plaintexts under the public key and test if they are equal to the ciphertext. A cryptosystem is called [semantically secure](#) if an attacker cannot distinguish two encryptions from each other even if the attacker knows (or has chosen) the corresponding plaintexts. As described above, RSA without padding is not semantically secure.<sup>[citation needed]</sup>
- RSA has the property that the product of two ciphertexts is equal to the encryption of the product of the respective plaintexts. That is  $m_1^e m_2^e \equiv (m_1 m_2)^e \pmod{n}$ . Because of this multiplicative property a [chosen-ciphertext attack](#) is possible. E.g., an attacker, who wants to know the decryption of a ciphertext  $c \equiv m^e \pmod{n}$  may ask the holder of the private key to decrypt an unsuspicious-looking ciphertext  $c' \equiv cr^e \pmod{n}$  for some value  $r$  chosen by the attacker. Because of the multiplicative property  $c'$  is the encryption of  $mr \pmod{n}$ . Hence, if the attacker is successful with the attack, he will learn  $mr \pmod{n}$  from which he can derive the message  $m$  by multiplying  $mr$  with the modular inverse of  $r$  modulo  $n$ .<sup>[citation needed]</sup>



## Padding schemes [\[edit\]](#)

To avoid these problems, practical RSA implementations typically embed some form of structured, randomized [padding](#) into the value  $m$  before encrypting it. This padding ensures that  $m$  does not fall into the range of insecure plaintexts, and that a given message, once padded, will encrypt to one of a large number of different possible ciphertexts.

Standards such as [PKCS#1](#) have been carefully designed to securely pad messages prior to RSA encryption. Because these schemes pad the plaintext  $m$  with some number of additional bits, the size of the un-padded message  $M$  must be somewhat smaller. RSA padding schemes must be carefully designed so as to prevent sophisticated attacks which may be facilitated by a predictable message structure. Early versions of the PKCS#1 standard (up to version 1.5) used a construction that appears to make RSA semantically secure. However, at Eurocrypt 2000, Coron et al. showed that for some types of messages, this padding does not provide a high enough level of security. Furthermore, at Crypto 1998, Bleichenbacher showed that this version is vulnerable to a practical [adaptive chosen ciphertext attack](#). Later versions of the standard include [Optimal Asymmetric Encryption Padding](#) (OAEP), which prevents these attacks. As such, OAEP should be used in any new application, and PKCS#1 v1.5 padding should be replaced wherever possible. The PKCS#1 standard also incorporates processing schemes designed to provide additional security for RSA signatures, e.g. the Probabilistic Signature Scheme for RSA ([RSA-PSS](#)).

Secure padding schemes such as RSA-PSS are as essential for the security of message signing as they are for message encryption. Two US patents on PSS were granted (USPTO 6266771 and USPTO 70360140); however, these patents expired on 24 July 2009 and 25 April 2010, respectively. Use of PSS no longer seems to be encumbered by patents. Note that using different RSA key-pairs for encryption and signing is potentially more secure.<sup>[14][15]</sup>

## Security and practical considerations [\[edit\]](#)

### Using the Chinese remainder algorithm [\[edit\]](#)

For efficiency many popular crypto libraries (like [OpenSSL](#), [Java](#) and [.NET](#)) use the following optimization for decryption and signing based on the [Chinese remainder theorem](#). The following values are precomputed and stored as part of the private key:

- $p$  and  $q$ : the primes from the key generation,
- $d_P = d \pmod{p-1}$ .
- $d_Q = d \pmod{q-1}$  and
- $q_{\text{inv}} = q^{-1} \pmod{p}$ .

These values allow the recipient to compute the exponentiation  $m = c^d \pmod{pq}$  more efficiently as follows:

- $m_1 = c^{d_P} \pmod{p}$
- $m_2 = c^{d_Q} \pmod{q}$
- $h = q_{\text{inv}}(m_1 - m_2) \pmod{p}$  (if  $m_1 < m_2$  then some libraries compute  $h$  as  $q_{\text{inv}}((m_1 + \lceil q/p \rceil p) - m_2) \pmod{p}$ )
- $m = m_2 + hq$

This is more efficient than computing  $m \equiv c^d \pmod{pq}$  even though two modular exponentiations have to be computed. The reason is that these two modular exponentiations both use a smaller exponent and a smaller modulus.

### Integer factorization and RSA problem [\[edit\]](#)

See also: [RSA Factoring Challenge](#), [Integer factorization records](#) and [Shor's algorithm](#)

The security of the RSA cryptosystem is based on two mathematical problems: the problem of [factoring large numbers](#) and the [RSA problem](#). Full decryption of an RSA ciphertext is thought to be infeasible on the assumption that both of these problems are hard, i.e., no efficient algorithm exists for solving them. Providing security against *partial* decryption may require the addition of a secure [padding scheme](#).<sup>[citation needed]</sup>

The [RSA problem](#) is defined as the task of taking  $e$ th roots modulo a composite  $n$ : recovering a value  $m$  such that  $c \equiv m^e \pmod{n}$ , where  $(n, e)$  is an RSA public key and  $c$  is an RSA ciphertext. Currently the most promising approach to solving the RSA problem is to factor the modulus  $n$ . With the ability to recover prime factors, an attacker can compute the secret exponent  $d$  from a public key  $(n, e)$ , then decrypt  $c$  using the standard procedure. To accomplish this, an attacker factors  $n$  into  $p$  and  $q$ , and computes  $(p-1)(q-1)$  which allows the

determination of  $d$  from  $e$ . No polynomial-time method for factoring large integers on a classical computer has yet been found, but it has not been proven that none exists. See [integer factorization](#) for a discussion of this problem.

Multiple polynomial quadratic sieve (MPQS) can be used to factor the public modulus  $n$ . The time taken to factor 128-bit and 256-bit  $n$  on a desktop computer (Processor: Intel Dual-Core i7-4500U 1.80GHz) are respectively 2 seconds and 35 minutes.

Bits	Time
128	Less than 2 seconds
192	16 seconds
256	35 minutes
260	1 hour

[16]

A tool called yafu can be used to optimize this process. The automation within YAFU is state-of-the-art, combining factorization algorithms in an intelligent and adaptive methodology that minimizes the time to find the factors of arbitrary input integers. Most algorithm implementations are multi-threaded, allowing YAFU to fully utilize multi- or many-core processors (including SNFS, GNFS, SIQS, and ECM). YAFU is primarily a command-line driven tool.<sup>[17]</sup> The time taken to factor  $n$  using yafu on the same computer was reduced to 103.1746 seconds. Yafu requires the GGNFS binaries to factor  $N$  that are 320 bits or larger. This is a very complicated software that requires a certain amount of technical skill to install and configure. It took about 5720s to factor 320bit- $N$  on the same computer.<sup>[16]</sup>

Bits	Time	Memory used
128	0.4886 seconds	0.1 MiB
192	3.9979 seconds	0.5 MiB
256	103.1746 seconds	3 MiB
300	1175.7826 seconds	10.9 MiB

[16]

In 2009, Benjamin Moody has factored an RSA-512 bit key in 73 days using only public software (GGNFS) and his desktop computer (dual-core Athlon64 at 1,900 MHz). Just under 5 gigabytes of disk was required and about 2.5 gigabytes of RAM for the sieving process. The first RSA-512 factorization in 1999 required the equivalent of 8,400 MIPS years over an elapsed time of about 7 months.<sup>[18]</sup>

Rivest, Shamir and Adleman note<sup>[2]</sup> that Miller has shown that – assuming the [Extended Riemann Hypothesis](#) – finding  $d$  from  $n$  and  $e$  is as hard as factoring  $n$  into  $p$  and  $q$  (up to a polynomial time difference).<sup>[19]</sup> However, Rivest, Shamir and Adleman note (in section IX / D of their paper) that they have not found a proof that inverting RSA is equally hard as factoring.

As of 2010, the largest factored [RSA number](#) was 768 bits long (232 decimal digits, see [RSA-768](#)). Its factorization, by a state-of-the-art distributed implementation, took around fifteen hundred CPU years (two years of real time, on many hundreds of computers). No larger RSA key is publically known to have been factored. In practice, RSA keys are typically 1024 to 4096 bits long. Some experts believe that 1024-bit keys may become breakable in the near future or may already be breakable by a sufficiently well-funded attacker (though this is disputed); few see any way that 4096-bit keys could be broken in the foreseeable future. Therefore, it is generally presumed that RSA is secure if  $n$  is sufficiently large. If  $n$  is 300 [bits](#) or shorter, it can be factored in a few hours on a [personal computer](#), using software already freely available. Keys of 512 bits have been shown to be practically breakable in 1999 when [RSA-155](#) was factored by using several hundred computers and are now factored in a few weeks using common hardware.<sup>[20]</sup> Exploits using 512-bit code-signing certificates that may have been factored were reported in 2011.<sup>[21]</sup> A theoretical hardware device named [TWIRL](#) and described by Shamir and Tromer in 2003 called into question the security of 1024 bit keys. It is currently recommended that  $n$  be at least 2048 bits long.<sup>[22]</sup>

In 1994, [Peter Shor](#) showed that a [quantum computer](#) (if one could ever be practically created for the purpose) would be able to factor in [polynomial time](#), breaking RSA; see [Shor's algorithm](#).

### Faulty key generation [\[edit\]](#)

See also: [Coppersmith's Attack](#) and [Wiener's Attack](#)

Finding the large primes  $p$  and  $q$  is usually done by testing random numbers of the right size with probabilistic [primality tests](#) which quickly eliminate virtually all non-primes.

Numbers  $p$  and  $q$  should not be 'too close', lest the [Fermat factorization](#) for  $n$  be successful, if  $p - q$ , for instance is less than  $2n^{1/4}$  (which for even small 1024-bit values of  $n$  is  $3 \times 10^{77}$ ) solving for  $p$  and  $q$  is trivial. Furthermore, if either  $p - 1$  or  $q - 1$  has only small prime factors,  $n$  can be factored quickly by [Pollard's  \$p - 1\$  algorithm](#), and these values of  $p$  or  $q$  should therefore be discarded as well.

It is important that the private key  $d$  be large enough. [Michael J. Wiener](#) showed<sup>[23]</sup> that if  $p$  is between  $q$  and  $2q$  (which is quite typical) and  $d < n^{1/4}/3$ , then  $d$  can be computed efficiently from  $n$  and  $e$ .

There is no known attack against small public exponents such as  $e = 3$ , provided that proper padding is used. [Coppersmith's Attack](#) has many applications in attacking RSA specifically if the public exponent  $e$  is small and if the encrypted message is short and not padded. [65537](#) is a commonly used value for  $e$ ; this value can be regarded as a compromise between avoiding potential small exponent attacks and still allowing efficient encryptions (or signature verification). The NIST Special Publication on Computer Security (SP 800-78 Rev 1 of August 2007) does not allow public exponents  $e$  smaller than 65537, but does not state a reason for this restriction.

### Importance of strong random number generation [\[edit\]](#)

A cryptographically strong [random number generator](#), which has been properly seeded with adequate entropy, must be used to generate the primes  $p$  and  $q$ . An analysis comparing millions of public keys gathered from the Internet was carried out in early 2012 by Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung and Christophe Wachter. They were able to factor 0.2% of the keys using only [Euclid's algorithm](#).<sup>[24][25]</sup>

They exploited a weakness unique to cryptosystems based on integer factorization. If  $n = pq$  is one public key and  $n' = p'q'$  is another, then if by chance  $p = p'$  (but  $q$  is not equal to  $q'$ ), then a simple computation of  $\gcd(n, n') = p$  factors both  $n$  and  $n'$ , totally compromising both keys. Lenstra et al. note that this problem can be minimized by using a strong random seed of bit-length twice the intended security level, or by employing a deterministic function to choose  $q$  given  $p$ , instead of choosing  $p$  and  $q$  independently.

[Nadia Heninger](#) was part of a group that did a similar experiment. They used an idea of [Daniel J. Bernstein](#) to compute the GCD of each RSA key  $n$  against the product of all the other keys  $n'$  they had found (a 729 million digit number), instead of computing each  $\gcd(n, n')$  separately, thereby achieving a very significant speedup since after one large division the GCD problem is of normal size.

Heninger says in her blog that the bad keys occurred almost entirely in embedded applications, including "firewalls, routers, VPN devices, remote server administration devices, printers, projectors, and VOIP phones" from over 30 manufacturers. Heninger explains that the one-shared-prime problem uncovered by the two groups results from situations where the pseudorandom number generator is poorly seeded initially and then reseeded between the generation of the first and second primes. Using seeds of sufficiently high entropy obtained from key stroke timings or electronic diode noise or [atmospheric noise](#) from a radio receiver tuned between stations should solve the problem.<sup>[26]</sup>

Strong random number generation is important throughout every phase of public key cryptography. For instance, if a weak generator is used for the symmetric keys that are being distributed by RSA, then an eavesdropper could bypass the RSA and guess the symmetric keys directly.

### Timing attacks [\[edit\]](#)

[Kocher](#) described a new attack on RSA in 1995: if the attacker Eve knows Alice's hardware in sufficient detail and is able to measure the decryption times for several known ciphertexts, she can deduce the decryption key  $d$  quickly. This attack can also be applied against the RSA signature scheme. In 2003, [Boneh](#) and [Brumley](#) demonstrated a more practical attack capable of recovering RSA factorizations over a network connection (e.g., from a [Secure Socket Layer](#) (SSL)-enabled webserver)<sup>[27]</sup> This attack takes advantage of information leaked by the [Chinese remainder theorem](#) optimization used by many RSA implementations.

One way to thwart these attacks is to ensure that the decryption operation takes a constant amount of time for every ciphertext. However, this approach can significantly reduce performance. Instead, most RSA implementations use an alternate technique known as [cryptographic blinding](#). RSA blinding makes use of the multiplicative property of RSA. Instead of computing  $c^d \pmod{n}$ , Alice first chooses a secret random value  $r$  and computes  $(r^e c)^d \pmod{n}$ . The result of this computation after applying [Euler's Theorem](#) is  $rc^d \pmod{n}$  and so the effect of  $r$  can be removed by multiplying by its inverse. A new value of  $r$  is chosen for each ciphertext. With blinding applied, the decryption time is no longer correlated to the value of the input ciphertext and so the timing



attack fails.

### Adaptive chosen ciphertext attacks [edit]

In 1998, [Daniel Bleichenbacher](#) described the first practical [adaptive chosen ciphertext attack](#), against RSA-encrypted messages using the PKCS #1 v1 [padding scheme](#) (a padding scheme randomizes and adds structure to an RSA-encrypted message, so it is possible to determine whether a decrypted message is valid). Due to flaws with the PKCS #1 scheme, Bleichenbacher was able to mount a practical attack against RSA implementations of the [Secure Socket Layer](#) protocol, and to recover session keys. As a result of this work, cryptographers now recommend the use of provably secure padding schemes such as [Optimal Asymmetric Encryption Padding](#), and RSA Laboratories has released new versions of PKCS #1 that are not vulnerable to these attacks.

### Side-channel analysis attacks [edit]

A side-channel attack using branch prediction analysis (BPA) has been described. Many processors use a [branch predictor](#) to determine whether a conditional branch in the instruction flow of a program is likely to be taken or not. Often these processors also implement [simultaneous multithreading](#) (SMT). Branch prediction analysis attacks use a spy process to discover (statistically) the private key when processed with these processors.

Simple Branch Prediction Analysis (SBPA) claims to improve BPA in a non-statistical way. In their paper, "On the Power of Simple Branch Prediction Analysis",<sup>[28]</sup> the authors of SBPA ([Onur Aciicmez](#) and [Cetin Kaya Koc](#)) claim to have discovered 508 out of 512 bits of an RSA key in 10 iterations.

A power fault attack on RSA implementations has been described in 2010.<sup>[29]</sup> The authors recovered the key by varying the CPU power voltage outside limits; this caused multiple power faults on the server.

### See also [edit]










































- [Trapdoor function](#)
- [Public-key cryptography](#)
- [Key exchange](#)
- [Diffie–Hellman key exchange](#)
- [Key management](#)
- [Cryptographic key length](#)
- [Computational complexity theory](#)
- [Acoustic cryptanalysis](#)

### Notes [edit]

- <sup>↑</sup> Namely, the values of  $m$  which are equal to  $-1$ ,  $0$ , or  $1$  modulo  $p$  while also equal to  $-1$ ,  $0$ , or  $1$  modulo  $q$ . There will be more values of  $m$  having  $c = m$  if  $p - 1$  or  $q - 1$  has other divisors in common with  $e - 1$  besides  $2$  because this gives more values of  $m$  such that  $m^{e-1} \pmod{p} = 1$  or  $m^{e-1} \pmod{q} = 1$  respectively.

### References [edit]

- <sup>↑</sup> Smart, Nigel (February 19, 2008). "Dr Clifford Cocks CB" . Bristol University. Retrieved August 14, 2011.
- <sup>^</sup> <sup>a</sup> <sup>b</sup> Rivest, R.; Shamir, A.; Adleman, L. (February 1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems"  (PDF). *Communications of the ACM* **21** (2): 120–126. doi:10.1145/359340.359342 .
- <sup>↑</sup> Diffie, W.; Hellman, M.E. (November 1976). "New directions in cryptography" . *IEEE Transactions on Information Theory* **22** (6): 644–654. doi:10.1109/TIT.1976.1055638 . ISSN 0018-9448 .
- <sup>↑</sup> Rivest, Ronald. "The Early Days of RSA – History and Lessons"  (PDF).
- <sup>↑</sup> Calderbank, Michael (2007-08-20). "The RSA Cryptosystem: History, Algorithm, Primes"  (PDF).
- <sup>^</sup> <sup>a</sup> <sup>b</sup> Robinson, Sara (June 2003). "Still Guarding Secrets after Years of Attacks, RSA Earns Accolades for its Founders"  (PDF). *SIAM News* **36** (5).
- <sup>↑</sup> [RSA Security Releases RSA Encryption Algorithm into Public Domain](#)  at the [Wayback Machine](#) (archived June 21, 2007)
- <sup>↑</sup> Boneh, Dan (1999). "Twenty Years of attacks on the RSA Cryptosystem" . *Notices of the American Mathematical Society* **46** (2): 203–213.
- <sup>↑</sup> "Further Attacks ON Server-Aided RSA Cryptosystems", James McKee and Richard Pinch, <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.1333>
- <sup>↑</sup> A Course in Number Theory and Cryptography, Graduate Texts in Math. No. 114, Springer-Verlag, New York, 1987. Neal Koblitz, Second edition, 1994. p94
- <sup>↑</sup> "common factors in (p-1) and (q-1)", Viktor Dukhovni, openssl-dev Digest, Vol 9, Issue 4, <https://www.mail->

- [archive.com/openssl-dev/%40openssl.org/msg39736.html](https://archive.com/openssl-dev/%40openssl.org/msg39736.html) , <https://www.mail-archive.com/openssl-dev/%40openssl.org/msg39725.html> 
12.  Hästad, Johan (1986). "On using RSA with Low Exponent in a Public Key Network". *Advances in Cryptology — CRYPTO '85 Proceedings*. Lecture Notes in Computer Science **218**. pp. 403–408. doi:10.1007/3-540-39799-X\_29 .
  13.  Coppersmith, Don (1997). "Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities". *Journal of Cryptology* **10** (4): 233–260. doi:10.1007/s001459900030 .
  14.  <http://stackoverflow.com/questions/5133246/what-is-the-purpose-of-using-separate-key-pairs-for-signing-and-encryption> .
  15.  [http://www.di-mgt.com.au/rsa\\_alg.html#weaknesses](http://www.di-mgt.com.au/rsa_alg.html#weaknesses) .
  16.     <http://iamnirosh.blogspot.com/2015/02/factoring-rsa-keys.html> .
  17.  <http://yafu.sourceforge.net/> .
  18.  <http://lukenotricks.blogspot.se/2009/08/solo-desktop-factorization-of-rsa-512.html> .
  19.  Gary L. Miller, "Riemann's Hypothesis and Tests for Primality" .
  20.  518-bit GNFS with msieve .
  21.  RSA-512 certificates abused in-the-wild .
  22.  Has the RSA algorithm been compromised as a result of Bernstein's Paper?  What key size should I be using?
  23.  Wiener, Michael J. (May 1990). "Cryptanalysis of short RSA secret exponents". *Information Theory, IEEE Transactions on* **36** (3): 553–558. doi:10.1109/18.54902 .
  24.  Markoff, John (February 14, 2012). "Flaw Found in an Online Encryption Method" . *New York Times*.
  25.  "Ron was wrong, Whit is right"  (PDF).
  26.  <https://freedom-to-tinker.com/blog/nadiah/new-research-theres-no-need-panic-over-factorable-keys-just-mind-your-ps-and-qs> .
  27.  Remote timing attacks are practical. . SSYM'03 Proceedings of the 12th conference on USENIX Security Symposium.
  28.  <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.80.1438&rep=rep1&type=pdf> .
  29.  FaultBased Attack of RSA Authentication 




## Further reading [\[edit\]](#)



---

- Menezes, Alfred; van Oorschot, Paul C.; Vanstone, Scott A. (October 1996). *Handbook of Applied Cryptography*[↗](#). CRC Press. ISBN 0-8493-8523-7.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). *Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill. pp. 881–887. ISBN 0-262-03293-7.

## External links [\[edit\]](#)

---

- The Original RSA Patent as filed with the U.S. Patent Office by Rivest; Ronald L. (Belmont, MA), Shamir; Adi (Cambridge, MA), Adleman; Leonard M. (Arlington, MA), December 14, 1977, **U.S. Patent 4,405,829**[↗](#).
- **PKCS #1: RSA Cryptography Standard**[↗](#) (RSA Laboratories website)
  - The *PKCS #1 standard* "provides recommendations for the implementation of *public-key cryptography* based on the **RSA** algorithm, covering the following aspects: cryptographic *primitives*; *encryption* schemes; *signature* schemes with appendix; *ASN.1* syntax for representing keys and for identifying the schemes".
- Explanation of RSA using colored lamps[↗](#) on YouTube
- Thorough walk through of RSA[↗](#)
- Prime Number Hide-And-Seek: How the RSA Cipher Works[↗](#)
- Onur Aciicmez, Cetin Kaya Koc, Jean-Pierre Seifert: *On the Power of Simple Branch Prediction Analysis*[↗](#)
- A New Vulnerability In RSA Cryptography, CAcert NEWS Blog[↗](#)
- Example of an RSA implementation with PKCS#1 padding (GPL source code)[↗](#)
- Kocher's article about timing attacks 
- An animated explanation of RSA with its mathematical background by CrypTool[↗](#)
- A spreadsheet implementing RSA[↗](#)
- *Hacking Secret Ciphers with Python*[↗](#), Chapter 24, Public Key Cryptography and the RSA Cipher[↗](#)
- Grime, James. "RSA Encryption"[↗](#). *Numberphile*. Brady Haran.

v · t · e	<b>Public-key cryptography</b>
<b>Algorithms</b>	AEDH · Benaloh · Blum–Goldwasser · Cayley–Purser · CEILIDH · Cramer–Shoup · Damgård–Jurik · DH · DSA · EPOC · ECDH · ECDSA · EdDSA · EKE · ElGamal (signature scheme) · <b>GMR</b> · Goldwasser–Micali · HFE · IES · Lamport · McEliece · Merkle–Hellman · MQV · Naccache–Stern · Naccache–Stern knapsack cryptosystem · NTRUEncrypt · NTRUSign · Paillier · Rabin · RSA · Okamoto–Uchiyama · Schnorr · Schmidt–Samoa · SPEKE · SRP · STS · Three-pass protocol · XTR
<b>Theory</b>	Discrete logarithm · Elliptic curve cryptography · Non-commutative cryptography · RSA problem
<b>Standardization</b>	CRYPTREC · IEEE P1363 · NESSIE · NSA Suite B
<b>Topics</b>	Digital signature · OAEP · Fingerprint · PKI · Web of trust · Key size
v · t · e	<b>Cryptography</b>
	History of cryptography · Cryptanalysis · Cryptography portal · Outline of cryptography
	Symmetric-key algorithm · Block cipher · Stream cipher · Public-key cryptography · Cryptographic hash function · Message authentication code · Random numbers · Steganography
	 <b>Mathematics portal</b>  <b>Cryptography portal</b>

Categories:
Public-key encryption schemes
|
Digital signature schemes
|
E-commerce