WIKIPEDIA
The Free Encyclopedia

WIKIPEDIA

RSA (cryptosystem)

RSA (**Rivest–Shamir–Adleman**) is a public-key cryptosystem, one of the oldest widely used for secure data transmission. The initialism "RSA" comes from the surnames of Ron Rivest, Adi Shamir and Leonard Adleman, who publicly described the algorithm in 1977. An equivalent system was developed secretly in 1973 at Government Communications Headquarters (GCHQ), the British signals intelligence agency, by the English mathematician Clifford Cocks. That system was declassified in 1997.[2]

In a public-key cryptosystem, the encryption key is public and distinct from the decryption key, which is kept secret (private). An RSA user creates and publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers are kept secret. Messages can be encrypted by anyone, via the public key, but can only be decoded by someone who knows the private key.[1]

The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem". Breaking RSA encryption is known as the RSA problem. Whether it is as difficult as the factoring problem is an open question.[3] There are no published methods to defeat the system if a large enough key is used.

RSA is a relatively slow algorithm. Because of this, it is not commonly used to directly encrypt user data. More often, RSA is used to transmit shared keys for symmetric-key cryptography, which are then used for bulk encryption–decryption.

RSA

General	
Designers	Ron Rivest,[1] Adi Shamir, and Leonard Adleman
First published	1977
Certification	PKCS#1, ANSI X9.31
Cipher detail	
Key sizes	2,048 to 4,096 bit typical
Rounds	1
Best public cryptanalysis	
General number field sieve for classical computers; Shor's algorithm for quantum computers. An 829-bit key has been broken.	

History

The idea of an asymmetric public-private key cryptosystem is attributed to Whitfield Diffie and Martin Hellman, who published this concept in 1976. They 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.[4]

Ron Rivest, Adi Shamir, and Leonard Adleman at the Massachusetts Institute of Technology made several attempts over the course of a year to create a function that was hard to invert. 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". For a time, they thought what they wanted to achieve was impossible due to contradictory requirements.[5] In April 1977, they spent Passover at the house of a student and drank a good deal of wine before returning to their homes at around midnight.[6] 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 he 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.[7]

Adi Shamir, co-inventor of RSA (the others are Ron Rivest and Leonard Adleman)

Clifford Cocks, an English mathematician working for the British intelligence agency Government Communications Headquarters (GCHQ), described a similar system in an internal document in 1973.[8] However, given the relatively expensive computers needed to implement it at the time, it was considered to be mostly a curiosity and, as far as is publicly known, was never deployed. His ideas and concepts, were not revealed until 1997 due to its top-secret classification.

Kid-RSA (KRSA) is a simplified, insecure public-key cipher published in 1997, designed for educational purposes. Some people feel that learning Kid-RSA gives insight into RSA and other public-key ciphers, analogous to simplified DES.[9][10][11][12][13]

Patent

A patent describing the RSA algorithm was granted to MIT on 20 September 1983: U.S. Patent 4,405,829 (https://patents.google.com/patent/US4405829) "Cryptographic communications system and method". From 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).

A detailed description of the algorithm was published in August 1977, in Scientific American's Mathematical Games column.[7] This preceded the patent's filing date of December 1977. Consequently, the patent had no legal standing outside the United States. Had Cocks' work been publicly known, a patent in the United States would not have been legal either.

When the patent was issued, terms of patent were 17 years. The patent was about to expire on 21 September 2000, but RSA Security released the algorithm to the public domain on 6 September 2000.[14]

Operation

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

A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d, and n, such that with modular exponentiation for all integers m (with $0 \leq m < n$):

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

and that knowing e and n, or even m, it can be extremely difficult to find d. Here the symbol ≡ denotes modular congruence: i.e. both $(m^e)^d$ and m have the same remainder when divided by n.

In addition, for some operations it is convenient that the order of the two exponentiations can be changed: the previous relation also implies

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

RSA involves a *public key* and a *private key*. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integers n and e, and the private key by the integer d (although n is also used during the decryption process, so it might be considered to be a part of the private key too). m represents the message (previously prepared with a certain technique explained below).

Key generation

The keys for the RSA algorithm are generated in the following way:

1. Choose two large prime numbers p and q.

- To make factoring harder, p and q should be chosen at random, be both large and have a large difference.[1] For choosing them the standard method is to choose random integers and use a primality test until two primes are found.
- p and q should be kept secret.

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.
   - n is released as part of the public key.
3. Compute $\lambda(n)$, where λ is Carmichael's totient function. Since $n = pq$, $\lambda(n) = \operatorname{lcm}(\lambda(p), \lambda(q))$, and since p and q are prime, $\lambda(p) = \varphi(p) = p - 1$, and likewise $\lambda(q) = q - 1$. Hence $\lambda(n) = \operatorname{lcm}(p - 1, q - 1)$.
   - The lcm may be calculated through the Euclidean algorithm, since $\operatorname{lcm}(a, b) = \frac{|ab|}{\gcd(a, b)}$.
   - $\lambda(n)$ is kept secret.
4. Choose an integer e such that $2 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; that is, e and $\lambda(n)$ are coprime.
   - e having a short bit-length and small Hamming weight results in more efficient encryption – the most commonly chosen value for e is $2^{16} + 1 = 65\,537$. The smallest (and fastest) possible value for e is 3, but such a small value for e has been shown to be less secure in some settings.[15]
   - e is released as part of the public key.
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; that is, d is the modular multiplicative inverse of e modulo $\lambda(n)$.
   - This means: solve for d the equation $de \equiv 1 \pmod{\lambda(n)}$; d can be computed efficiently by using the extended Euclidean algorithm, since, thanks to e and $\lambda(n)$ being coprime, said equation is a form of Bézout's identity, where d is one of the coefficients.
   - d is kept secret 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 private (or decryption) exponent d, which must be kept secret. p, q, and $\lambda(n)$ must also be kept secret because they can be used to calculate d. In fact, they can all be discarded after d has been computed.[16]

In the original RSA paper,[1] the Euler totient function $\varphi(n) = (p - 1)(q - 1)$ is used instead of $\lambda(n)$ for calculating the private exponent d. Since $\varphi(n)$ is always divisible by $\lambda(n)$, the algorithm works as well. The possibility of using Euler totient function results also from Lagrange's theorem applied to the multiplicative group of integers modulo pq. Thus any d satisfying $d \cdot e \equiv 1 \pmod{\varphi(n)}$ also satisfies $d \cdot e \equiv 1 \pmod{\lambda(n)}$. However, computing d modulo $\varphi(n)$ will sometimes yield a result that is larger than necessary (i.e. $d > \lambda(n)$). Most of the implementations of RSA will accept exponents generated using either method (if they use the private exponent d at all, rather than using the optimized decryption method based on the Chinese remainder theorem described below), but some standards such as FIPS 186-4 (https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf#page=63) (Section B.3.1) may require that $d < \lambda(n)$. Any "oversized" private exponents not meeting this criterion may always be reduced modulo $\lambda(n)$ to obtain a smaller equivalent exponent.

Since any common factors of $(p-1)$ and $(q-1)$ are present in the factorisation of $n - 1 = pq - 1 = (p-1)(q-1) + (p-1) + (q-1)$,[17] it is recommended that $(p-1)$ and $(q-1)$ have only very small common factors, if any, besides the necessary 2.[1][18][19][20]

Note: The authors of the original RSA paper carry out the key generation by choosing d and then computing e as the modular multiplicative inverse of d modulo $\varphi(n)$, whereas most current implementations of RSA, such as those following PKCS#1, do the reverse (choose e and compute d). Since the chosen key can be small, whereas the computed key normally is not, the RSA paper's algorithm optimizes decryption compared to encryption, while the modern algorithm optimizes encryption instead.[1][21]

Key distribution

Suppose that Bob wants to send information to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message, and Alice must use her private key to decrypt the message.

To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable, but not necessarily secret, route. Alice's private key (d) is never distributed.

Encryption

After Bob obtains Alice's public key, he can send a message M to Alice.

To do it, he first turns M (strictly speaking, the un-padded plaintext) into an integer m (strictly speaking, the padded plaintext), such that $0 \leq m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext c, using Alice's public key e, corresponding to

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

This can be done reasonably quickly, even for very large 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,[a] but this is very unlikely to occur in practice.

Decryption

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

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

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

Example

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.

1. Choose two distinct prime numbers, such as

   $p = 61$ and $q = 53$.

2. Compute $n = pq$ giving

   $n = 61 \times 53 = 3233.$

3. Compute the Carmichael's totient function of the product as $\lambda(n) = \operatorname{lcm}(p-1, q-1)$ giving

   $\lambda(3233) = \operatorname{lcm}(60, 52) = 780.$

4. Choose any number $2 < e < 780$ that is coprime to 780. Choosing a prime number for e leaves us only to check that e is not a divisor of 780.

   Let $e = 17$.

5. Compute d, the modular multiplicative inverse of $e \pmod{\lambda(n)}$, yielding

$$d = 413,$$

$$\text{as } 1 = (17 \times 413) \bmod 780.$$

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

$$\begin{aligned} c(m) &= m^e \bmod n \\ &= m^{17} \bmod 3233. \end{aligned}$$

The **private key** is (n = 3233, d = 413). For an encrypted ciphertext c, the decryption function is

$$\begin{aligned} m(c) &= c^d \bmod n \\ &= c^{413} \bmod 3233. \end{aligned}$$

For instance, in order to encrypt m = 65, one calculates

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

To decrypt c = 2790, one calculates

$$m = 2790^{413} \bmod 3233 = 65.$$

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. e, also from the public key, is then inverted to get d, thus acquiring the private key.

Practical implementations use the Chinese remainder theorem to speed up the calculation using modulus of factors (mod pq using mod p and mod q).

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

$$\begin{aligned} d_p &= d \bmod (p-1) = 413 \bmod (61-1) = 53, \\ d_q &= d \bmod (q-1) = 413 \bmod (53-1) = 49, \\ q_{\text{inv}} &= q^{-1} \bmod p = 53^{-1} \bmod 61 = 38 \\ &\Rightarrow (q_{\text{inv}} \times q) \bmod p = 38 \times 53 \bmod 61 = 1. \end{aligned}$$

Here is how d_p, d_q and q_{inv} are used for efficient decryption (encryption is efficient by choice of a suitable d and e pair):

$$\begin{aligned} m_1 &= c^{d_p} \bmod p = 2790^{53} \bmod 61 = 4, \\ m_2 &= c^{d_q} \bmod q = 2790^{49} \bmod 53 = 12, \\ h &= (q_{\text{inv}} \times (m_1 - m_2)) \bmod p = (38 \times -8) \bmod 61 = 1, \\ m &= m_2 + h \times q = 12 + 1 \times 53 = 65. \end{aligned}$$

Signing messages

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 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 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 being sent.

This works because of exponentiation rules:

$$h = \operatorname{hash}(m),$$

$$(h^e)^d = h^{ed} = h^{de} = (h^d)^e \equiv h \pmod{n}.$$

Thus the keys may be swapped without loss of generality, that is, a private key of a key pair may be used either to:

1. Decrypt a message only intended for the recipient, which may be encrypted by anyone having the public key (asymmetric encrypted transport).
2. Encrypt a message which may be decrypted by anyone, but which can only be encrypted by one person; this provides a digital signature.

Proofs of correctness

Proof using Fermat's little theorem

The proof of the correctness of RSA is based on Fermat's little theorem, stating that $a^{p-1} \equiv 1 \pmod{p}$ for any integer a and prime p, not dividing a.[note 1]

We want to show that

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

for every integer m when p and q are distinct prime numbers and e and d are positive integers satisfying $ed \equiv 1 \pmod{\lambda(pq)}$.

Since $\lambda(pq) = \operatorname{lcm}(p-1, q-1)$ is, by construction, divisible by both $p-1$ and $q-1$, we can write

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

for some nonnegative integers h and k.[note 2]

To check whether two numbers, such as m^{ed} and m, are congruent mod pq, it suffices (and in fact is equivalent) to check that they are congruent mod p and mod q separately.[note 3]

To show $m^{ed} \equiv m \pmod{p}$, we consider two cases:

1. If $m \equiv 0 \pmod{p}$, m is a multiple of p. Thus m^{ed} is a multiple of p. So $m^{ed} \equiv 0 \equiv m \pmod{p}$.
2. If $m \not\equiv 0 \pmod{p}$,

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

where we used Fermat's little theorem to replace $m^{p-1} \bmod p$ with 1.

The verification that $m^{ed} \equiv m \pmod{q}$ proceeds in a completely analogous way:

1. If $m \equiv 0 \pmod{q}$, m^{ed} is a multiple of q. So $m^{ed} \equiv 0 \equiv m \pmod{q}$.
2. If $m \not\equiv 0 \pmod{q}$,

$$m^{ed} = m^{ed-1}m = m^{k(q-1)}m = (m^{q-1})^k m \equiv 1^k m \equiv m \pmod{q}.$$

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

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

Notes

1. We cannot trivially break RSA by applying the theorem (mod *pq*) because pq is not prime.
2. In particular, the statement above holds for any e and d that satisfy $ed \equiv 1 \pmod{(p-1)(q-1)}$, since $(p-1)(q-1)$ is divisible by $\lambda(pq)$, and thus trivially also by $p-1$ and $q-1$. However, in modern implementations of RSA, it is common to use a reduced private exponent d that only satisfies the weaker, but sufficient condition $ed \equiv 1 \pmod{\lambda(pq)}$.
3. This is part of the Chinese remainder theorem, although it is not the significant part of that theorem.

Proof using Euler's theorem

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 $ed \equiv 1 \pmod{\varphi(n)}$. Since e and d are positive, we can write $ed = 1 + h\varphi(n)$ for some non-negative integer h. *Assuming* that m is relatively prime to n, we have

$$m^{ed} = m^{1+h\varphi(n)} = m(m^{\varphi(n)})^h \equiv m(1)^h \equiv m \pmod{n},$$

where the second-last congruence follows from Euler's theorem.

More generally, for any e and d satisfying $ed \equiv 1 \pmod{\lambda(n)}$, the same conclusion follows from Carmichael's generalization of Euler's theorem, which states that $m^{\lambda(\mathrm{n})} \equiv 1 \pmod{n}$ for all m relatively prime to n.

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

Attacks against plain RSA

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 decrypted easily by taking the eth 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 clear texts are not equal, but the attacker knows a linear relation between them.[22] This attack was later improved by Don Coppersmith (see Coppersmith's attack).[23]
- 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 whether 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. RSA without padding is not semantically secure.[24]
- 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 d 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, they will learn $mr \pmod{n}$, from which they can derive the message m by multiplying mr with the modular inverse of r modulo n.

- Given the private exponent d, one can efficiently factor the modulus $n = pq$. And given factorization of the modulus $n = pq$, one can obtain any private key (d', n) generated against a public key (e', n).[15]

Padding schemes

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 that 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 Crypto 1998, Bleichenbacher showed that this version is vulnerable to a practical adaptive chosen-ciphertext attack. Furthermore, at Eurocrypt 2000, Coron et al.[25] showed that for some types of messages, this padding does not provide a high enough level of security. 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 USA patents on PSS were granted (U.S. Patent 6,266,771 (https://patents.google.com/patent/US6266771) and U.S. Patent 7,036,014 (https://patents.google.com/patent/US7036014)); 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.[26]

Security and practical considerations

Using the Chinese remainder algorithm

For efficiency, many popular crypto libraries (such as OpenSSL, Java and .NET) use for decryption and signing the following optimization 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,

- $d_P = d \pmod{p-1}$,
- $d_Q = d \pmod{q-1}$,
- $q_{\text{inv}} = q^{-1} \pmod{p}$.

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

$$m_1 = c^{d_P} \pmod{p},$$
$$m_2 = c^{d_Q} \pmod{q},$$
$$h = q_{\text{inv}}(m_1 - m_2) \pmod{p},$$[b]
$$m = m_2 + hq \pmod{pq}\ .$$

This is more efficient than computing exponentiation by squaring, 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 the RSA problem

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.[27]

The RSA problem is defined as the task of taking eth 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 $\operatorname{lcm}(p-1, q-1)$ that 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 first RSA-512 factorization in 1999 used hundreds of computers and required the equivalent of 8,400 MIPS years, over an elapsed time of about seven months.[28] By 2009, Benjamin Moody could factor an 512-bit RSA key in 73 days using only public software (GGNFS) and his desktop computer (a dual-core Athlon64 with a 1,900 MHz CPU). Just less than 5 gigabytes of disk storage was required and about 2.5 gigabytes of RAM for the sieving process.

Rivest, Shamir, and Adleman noted[1] that Miller has shown that – assuming the truth of 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).[29] However, Rivest, Shamir, and Adleman noted, in section IX/D of their paper, that they had not found a proof that inverting RSA is as hard as factoring.

As of 2020, the largest publicly known factored RSA number had 829 bits (250 decimal digits, RSA-250).[30] Its factorization, by a state-of-the-art distributed implementation, took about 2,700 CPU-years. In practice, RSA keys are typically 1024 to 4096 bits long. In 2003, RSA Security estimated that 1024-bit keys were likely to become crackable by 2010.[31] As of 2020, it is not known whether such keys can be cracked, but minimum recommendations have moved to at least 2048 bits.[32] It is generally presumed that RSA is secure if n is sufficiently large, outside of quantum computing.

If n is 300 bits or shorter, it can be factored in a few hours in 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 these are now factored in a few weeks using common hardware. Exploits using 512-bit code-signing certificates that may have been factored were reported in 2011.[33] A theoretical hardware device named TWIRL, described by Shamir and Tromer in 2003, called into question the security of 1024-bit keys.[31]

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

Finding the large primes p and q is usually done by testing random numbers of the correct size with probabilistic primality tests that quickly eliminate virtually all of the nonprimes.

The numbers p and q should not be "too close", lest the Fermat factorization for n be successful. If $p - q$ is less than $2n^{1/4}$ ($n = p \cdot q$, which even for "small" 1024-bit values of n is 3×10^{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 hence such values of p or q should be discarded.

It is important that the private exponent d be large enough. Michael J. Wiener showed 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.[34]

There is no known attack against small public exponents such as $e = 3$, provided that the 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.

In October 2017, a team of researchers from Masaryk University announced the ROCA vulnerability, which affects RSA keys generated by an algorithm embodied in a library from Infineon known as RSALib. A large number of smart cards and trusted platform modules (TPM) were shown to be affected. Vulnerable RSA keys are easily identified using a test program the team released.[35]

Importance of strong random number generation

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.[36][37]

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 more than 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 is 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.[38]

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 RSA and guess the symmetric keys directly.

Timing attacks

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, Eve 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 Sockets Layer (SSL)-enabled webserver).[39] 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

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 Sockets 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.

A variant of this attack, dubbed "BERserk", came back in 2014.[40][41] It impacted the Mozilla NSS Crypto Library, which was used notably by Firefox and Chrome.

Side-channel analysis attacks

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",[42] 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 was described in 2010.[43] The author recovered the key by varying the CPU power voltage outside limits; this caused multiple power faults on the server.

Tricky implementation

There are many details to keep in mind in order to implement RSA securely (strong PRNG, acceptable public exponent, etc.). This makes the implementation challenging, to the point the book Practical Cryptography With Go suggests avoiding RSA if possible.[44]

Implementations

Some cryptography libraries that provide support for RSA include:

- Botan
- Bouncy Castle
- cryptlib
- Crypto++
- Libgcrypt
- Nettle
- OpenSSL
- wolfCrypt
- GnuTLS
- mbed TLS
- LibreSSL

See also

- Acoustic cryptanalysis
- Computational complexity theory
- Diffie–Hellman key exchange
- Digital Signature Algorithm
- Elliptic-curve cryptography
- Key exchange
- Key management

Mathematics portal

- Key size
- Public-key cryptography
- Trapdoor function

Notes

a. 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 $m^{e-1} \bmod p = 1$ or $m^{e-1} \bmod q = 1$ respectively.

b. If $m_1 < m_2$, then some libraries compute h as $q_{\text{inv}} \left[\left(m_1 + \left\lceil \frac{q}{p} \right\rceil p\right) - m_2\right] \pmod p$.

References

1. Rivest, R.; Shamir, A.; Adleman, L. (February 1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems" (https://web.archive.org/web/20230127011251/http://people.csail.mit.edu/rivest/Rsapaper.pdf) (PDF). *Communications of the ACM*. **21** (2): 120–126. CiteSeerX 10.1.1.607.2677 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.607.2677). doi:10.1145/359340.359342 (https://doi.org/10.1145%2F359340.359342). S2CID 2873616 (https://api.semanticscholar.org/CorpusID:2873616). Archived from the original (http://people.csail.mit.edu/rivest/Rsapaper.pdf) (PDF) on 2023-01-27.
2. Smart, Nigel (February 19, 2008). "Dr Clifford Cocks CB" (http://www.bristol.ac.uk/graduation/honorary-degrees/hondeg08/cocks.html). Bristol University. Retrieved August 14, 2011.
3. Castelvecchi, Davide (2020-10-30). "Quantum-computing pioneer warns of complacency over Internet security" (https://www.nature.com/articles/d41586-020-03068-9). *Nature*. **587** (7833): 189. Bibcode:2020Natur.587..189C (https://ui.adsabs.harvard.edu/abs/2020Natur.587..189C). doi:10.1038/d41586-020-03068-9 (https://doi.org/10.1038%2Fd41586-020-03068-9). PMID 33139910 (https://pubmed.ncbi.nlm.nih.gov/33139910). S2CID 226243008 (https://api.semanticscholar.org/CorpusID:226243008). 2020 interview of Peter Shor.
4. Diffie, W.; Hellman, M. E. (November 1976). "New directions in cryptography". *IEEE Transactions on Information Theory*. **22** (6): 644–654. CiteSeerX 10.1.1.37.9720 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.9720). doi:10.1109/TIT.1976.1055638 (https://doi.org/10.1109%2FTIT.1976.1055638). ISSN 0018-9448 (https://www.worldcat.org/issn/0018-9448).
5. Rivest, Ronald. "The Early Days of RSA – History and Lessons" (https://people.csail.mit.edu/rivest/pubs/ARS03.rivest-slides.pdf) (PDF).

6. Calderbank, Michael (2007-08-20). "The RSA Cryptosystem: History, Algorithm, Primes" (http://www.math.uchicago.edu/~may/VIGRE/VIGRE2007/REUPapers/FINALAPP/Calderbank.pdf) (PDF).
7. Robinson, Sara (June 2003). "Still Guarding Secrets after Years of Attacks, RSA Earns Accolades for its Founders" (http://www.msri.org/people/members/sara/articles/rsa.pdf) (PDF). *SIAM News*. **36** (5).
8. Cocks, C. C. (20 November 1973). "A Note on Non-Secret Encryption" (https://web.archive.org/web/20180928121748/https://www.gchq.gov.uk/sites/default/files/document_files/Cliff%20Cocks%20paper%2019731120.pdf) (PDF). *www.gchq.gov.uk*. Archived from the original (https://www.gchq.gov.uk/sites/default/files/document_files/Cliff%20Cocks%20paper%2019731120.pdf) (PDF) on 28 September 2018. Retrieved 2017-05-30.
9. Jim Sauerberg. "From Private to Public Key Ciphers in Three Easy Steps" (https://ww2.amstat.org/mam/06/Sauerberg_PKC-essay.html).
10. Margaret Cozzens and Steven J. Miller. "The Mathematics of Encryption: An Elementary Introduction" (https://books.google.com/books?id=GbKyAAAAQBAJ). p. 180.
11. Alasdair McAndrew. "Introduction to Cryptography with Open-Source Software" (https://books.google.com/books?id=9lTRBQAAQBAJ). p. 12.
12. Surender R. Chiluka. "Public key Cryptography" (https://web.archive.org/web/20220319203917/https://www.cs.uri.edu/cryptography/publickeykidkrypto.htm).
13. Neal Koblitz. "Cryptography As a Teaching Tool" (https://sites.math.washington.edu/~koblitz/crlogia.html). Cryptologia, Vol. 21, No. 4 (1997).
14. "RSA Security Releases RSA Encryption Algorithm into Public Domain" (https://web.archive.org/web/20070621021111/http://www.rsa.com/press_release.aspx?id=261). Archived from the original (http://www.rsa.com/press_release.aspx?id=261) on June 21, 2007. Retrieved 2010-03-03.
15. Boneh, Dan (1999). "Twenty Years of attacks on the RSA Cryptosystem" (http://crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html). *Notices of the American Mathematical Society*. **46** (2): 203–213.
16. Applied Cryptography, John Wiley & Sons, New York, 1996. Bruce Schneier, p. 467.
17. McKee, James; Pinch, Richard (1998). "Further Attacks on Server-Aided RSA Cryptosystems" (https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=64294c404088b69a519614510b8d12b4809a6b10). CiteSeerX 10.1.1.33.1333 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.1333).
18. A Course in Number Theory and Cryptography, Graduate Texts in Math. No. 114, Springer-Verlag, New York, 1987. Neal Koblitz, Second edition, 1994. p. 94.
19. Dukhovni, Viktor (July 31, 2015). "common factors in (*p* – 1) and (*q* – 1)" (https://mta.openssl.org/pipermail/openssl-dev/2015-July/002266.html). *openssl-dev* (Mailing list).
20. Dukhovni, Viktor (August 1, 2015). "common factors in (*p* – 1) and (*q* – 1)" (https://mta.openssl.org/pipermail/openssl-dev/2015-August/002277.html). *openssl-dev* (Mailing list).
21. Johnson, J.; Kaliski, B. (February 2003). *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1* (https://datatracker.ietf.org/doc/html/rfc3447). Network Working Group. doi:10.17487/RFC3447 (https://doi.org/10.17487%2FRFC3447). RFC 3447 (https://datatracker.ietf.org/doc/html/rfc3447). Retrieved 9 March 2016.
22. Håstad, Johan (1986). "On using RSA with Low Exponent in a Public Key Network". *Advances in Cryptology – CRYPTO '85 Proceedings*. Lecture Notes in Computer Science. Vol. 218. pp. 403–408. doi:10.1007/3-540-39799-X_29 (https://doi.org/10.1007%2F3-540-39799-X_29). ISBN 978-3-540-16463-0.

23. Coppersmith, Don (1997). "Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities" (https://www.di.ens.fr/~fouque/ens-rennes/coppersmith.pdf) (PDF). *Journal of Cryptology*. **10** (4): 233–260. CiteSeerX 10.1.1.298.4806 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.298.4806). doi:10.1007/s001459900030 (https://doi.org/10.1007%2Fs001459900030). S2CID 15726802 (https://api.semanticscholar.org/CorpusID:15726802).
24. Goldwasser, Shafi; Micali, Silvio (1982-05-05). "Probabilistic encryption & how to play mental poker keeping secret all partial information" (https://doi.org/10.1145/800070.802212). *Proceedings of the fourteenth annual ACM symposium on Theory of computing - STOC '82*. New York, NY, USA: Association for Computing Machinery. pp. 365–377. doi:10.1145/800070.802212 (https://doi.org/10.1145%2F800070.802212). ISBN 978-0-89791-070-5. S2CID 10316867 (https://api.semanticscholar.org/CorpusID:10316867).
25. Coron, Jean-Sébastien; Joye, Marc; Naccache, David; Paillier, Pascal (2000). "New Attacks on PKCS#1 v1.5 Encryption". In Preneel, Bart (ed.). *Advances in Cryptology — EUROCRYPT 2000*. Lecture Notes in Computer Science. Vol. 1807. Berlin, Heidelberg: Springer. pp. 369–381. doi:10.1007/3-540-45539-6_25 (https://doi.org/10.1007%2F3-540-45539-6_25). ISBN 978-3-540-45539-4.
26. "RSA Algorithm" (https://www.di-mgt.com.au/rsa_alg.html#weaknesses).
27. Machie, Edmond K. (29 March 2013). *Network security traceback attack and react in the United States Department of Defense network* (https://books.google.com/books?id=AK5MySZbbuMC&pg=PA167). Trafford. p. 167. ISBN 978-1466985742.
28. Lenstra, Arjen; et al. (Group) (2000). "Factorization of a 512-bit RSA Modulus" (https://www.iacr.org/archive/eurocrypt2000/1807/18070001-new.pdf) (PDF). Eurocrypt.
29. Miller, Gary L. (1975). "Riemann's Hypothesis and Tests for Primality" (https://www.cs.cmu.edu/~glmiller/Publications/Papers/Mi75.pdf) (PDF). *Proceedings of Seventh Annual ACM Symposium on Theory of Computing*. pp. 234–239.
30. Zimmermann, Paul (2020-02-28). "Factorization of RSA-250" (https://web.archive.org/web/20200228234716/https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html). Cado-nfs-discuss. Archived from the original (https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2020-February/001166.html) on 2020-02-28. Retrieved 2020-07-12.
31. Kaliski, Burt (2003-05-06). "TWIRL and RSA Key Size" (https://web.archive.org/web/20170417095741/https://www.emc.com/emc-plus/rsa-labs/historical/twirl-and-rsa-key-size.htm). RSA Laboratories. Archived from the original (http://emc.com/emc-plus/rsa-labs/historical/twirl-and-rsa-key-size.htm) on 2017-04-17. Retrieved 2017-11-24.
32. Barker, Elaine; Dang, Quynh (2015-01-22). "NIST Special Publication 800-57 Part 3 Revision 1: Recommendation for Key Management: Application-Specific Key Management Guidance" (http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf) (PDF). National Institute of Standards and Technology. p. 12. doi:10.6028/NIST.SP.800-57pt3r1 (https://doi.org/10.6028%2FNIST.SP.800-57pt3r1). Retrieved 2017-11-24.
33. Sandee, Michael (November 21, 2011). "RSA-512 certificates abused in-the-wild" (https://blog.fox-it.com/2011/11/21/rsa-512-certificates-abused-in-the-wild/). *Fox-IT International blog*.
34. Wiener, Michael J. (May 1990). "Cryptanalysis of short RSA secret exponents" (http://www.cits.rub.de/imperia/md/content/may/krypto2ss08/shortsecretexponents.pdf) (PDF). *IEEE Transactions on Information Theory*. **36** (3): 553–558. doi:10.1109/18.54902 (https://doi.org/10.1109%2F18.54902). S2CID 7120331 (https://api.semanticscholar.org/CorpusID:7120331).
35. Nemec, Matus; Sys, Marek; Svenda, Petr; Klinec, Dusan; Matyas, Vashek (November 2017). "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli" (https://crocs.fi.muni.cz/_media/public/papers/nemec_roca_ccs17_preprint.pdf) (PDF). *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. doi:10.1145/3133956.3133969 (https://doi.org/10.1145%2F3133956.3133969).

36. Markoff, John (February 14, 2012). "Flaw Found in an Online Encryption Method" (https://www.nytimes.com/2012/02/15/technology/researchers-find-flaw-in-an-online-encryption-method.html). *The New York Times*.
37. Lenstra, Arjen K.; Hughes, James P.; Augier, Maxime; Bos, Joppe W.; Kleinjung, Thorsten; Wachter, Christophe (2012). "Ron was wrong, Whit is right" (http://eprint.iacr.org/2012/064.pdf) (PDF).
38. Heninger, Nadia (February 15, 2012). "New research: There's no need to panic over factorable keys–just mind your Ps and Qs" (https://freedom-to-tinker.com/blog/nadiah/new-research-theres-no-need-panic-over-factorable-keys-just-mind-your-ps-and-qs). *Freedom to Tinker*.
39. Brumley, David; Boneh, Dan (2003). "Remote timing attacks are practical" (http://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf) (PDF). *Proceedings of the 12th Conference on USENIX Security Symposium*. SSYM'03.
40. " 'BERserk' Bug Uncovered In Mozilla NSS Crypto Library Impacts Firefox, Chrome" (https://www.darkreading.com/attacks-breaches/-berserk-bug-uncovered-in-mozilla-nss-crypto-library-impacts-firefox-chrome). 25 September 2014. Retrieved 4 January 2022.
41. "RSA Signature Forgery in NSS" (https://www.mozilla.org/en-US/security/advisories/mfsa2014-73/). *Mozilla*.
42. Acıiçmez, Onur; Koç, Çetin Kaya; Seifert, Jean-Pierre (2007). "On the power of simple branch prediction analysis". *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security*. ASIACCS '07. pp. 312–320. CiteSeerX 10.1.1.80.1438 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.80.1438). doi:10.1145/1229285.1266999 (https://doi.org/10.1145%2F1229285.1266999).
43. Pellegrini, Andrea; Bertacco, Valeria; Austin, Todd (2010). "Fault-Based Attack of RSA Authentication" (https://www.eecs.umich.edu/~valeria/research/publications/DATE10RSA.pdf) (PDF).
44. Isom, Kyle. "Practical Cryptography With Go" (https://leanpub.com/gocrypto/read#leanpub-auto-rsa). Retrieved 4 January 2022.

Further reading

- Menezes, Alfred; van Oorschot, Paul C.; Vanstone, Scott A. (October 1996). *Handbook of Applied Cryptography* (https://archive.org/details/handbookofapplie0000mene). CRC Press. ISBN 978-0-8493-8523-0.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). *Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill. pp. 881 (https://archive.org/details/introductiontoal00corm_691/page/n903)–887. ISBN 978-0-262-03293-3.

External links

- 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 (https://patents.google.com/patent/US4405829)**.
- PKCS #1: RSA Cryptography Standard (http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/pkcs-rsa-cryptography-standard.htm) (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 (https://www.youtube.com/watch?v=vgTtHV04xRI) on YouTube
- Thorough walk through of RSA (http://www.di-mgt.com.au/rsa_alg.html)
- Prime Number Hide-And-Seek: How the RSA Cipher Works (http://www.muppetlabs.com/~breadbox/txt/rsa.html)
- Onur Aciicmez, Cetin Kaya Koc, Jean-Pierre Seifert: *On the Power of Simple Branch Prediction Analysis* (http://eprint.iacr.org/2006/351)
- Example of an RSA implementation with PKCS#1 padding (GPL source code) (http://polarssl.org/source_code) Archived (https://web.archive.org/web/20120722193444/http://polarssl.org/source_code) 2012-07-22 at the Wayback Machine
- Kocher's article about timing attacks (http://www.cryptography.com/resources/whitepapers/TimingAttacks.pdf)
- An animated explanation of RSA with its mathematical background by CrypTool (https://www.cryptool.org/assets/ct1/presentations/RSA/RSA-en.pptx)
- Grime, James. "RSA Encryption" (https://web.archive.org/web/20181006042830/http://www.numberphile.com/videos/RSA.html). *Numberphile*. Brady Haran. Archived from the original (http://www.numberphile.com/videos/RSA.html) on 2018-10-06. Retrieved 2013-04-13.
- How RSA Key used for Encryption in real world (http://www.gnudeveloper.com/groups/cyber_security/Cryptography_RSA_Key_Exchange_works_in_realtime_using_Keytool_openSSL%20.html)

Retrieved from "https://en.wikipedia.org/w/index.php?title=RSA_(cryptosystem)&oldid=1193042466"

-