The RSA Algorithm

Amanda

September 29, 2020

Contents

Introduction

The Rivest-Shamir-Adleman (RSA) Algorithm, published in 1977, is named after its creators, Ron Rivest, Adi Shamir, and Leonard Adleman. It is the first implementation of the public key cryptosystem and is still used in in-transit encryption and digital signatures today [Aum17]. In this essay, I will first discuss the mathematics behind the algorithm, its security based on the factoring problem, and its implementation. This will be followed by how the algorithm is augmented to meet the security goals in practice, and some attacks against RSA-based cryptosystems.

The Algorithm and the Underlying Mathematics

The Public Key Cryptosystem

The public key cryptosystem invented by Diffie and Hellman describes an encryption procedure E as a function. The domain and the codomain of E are the same set M, where M is a finite set of values that a message can take. The decryption procedure D is the inverse of E. Each procedure has a particular key associated with it. For $m \in M$, it should be efficient to compute both $E(m)$ and $D(E(m)) = m$, but infeasible to deduce D or m from E and $E(m)$ alone [DH76].

Such a function E is called a "trapdoor one-way permutation" [RSA78]. E is a permutation because it is invertible and hence bijective, and its codomain is the same as its domain [HO00]. E is one-way due to the difficulty of computing m from $E(m)$ even when E is known. Nevertheless, the computation becomes feasible if the "trapdoor information" which reveals D is known [DH76].

The RSA Algorithm

The RSA algorithm is proposed in [RSA78]. It works on numbers in $\mathbb{Z}_n$, using modular arithmetic with n as the modulus, satisfying

$$n = p \cdot q \tag{1}$$

where p and q are two distinct large prime numbers chosen randomly. $\mathbb{Z}_n$ corresponds to the set M described above.
Let e and d be positive integers less than n such that

$$e \cdot d \equiv 1 \pmod{((p-1)(q-1))} \tag{2}$$

(e, n) is the encryption key, and (d, n) is the decryption key. d is also called the secret exponent [Aum17] as it is the trapdoor information that makes decryption efficient.
The encryption procedure E is implemented by

$$c = E(m) = m^e \bmod n \tag{3}$$

The decryption procedure D is implemented by

$$m = D(c) = c^d \bmod n \tag{4}$$

where $m \in \mathbb{Z}_n$ denotes the message in plaintext and c denotes the corresponding ciphertext.

The Mathematics in Encryption and Decryption

The following proof that D is the inverse of E is due to a lecture by Dijk [DL].

Euler's totient function Euler's totient function, ϕ, takes a positive integer n and returns the number of positive integers in $\{1, 2, ..., n-1\}$ which are co-prime with n.

Euler's theorem If $\gcd(n, k) = 1$, then $k^{\phi(n)} \equiv 1 \pmod{n}$. ($\gcd(n, k)$ is the greatest common divisor of n and k.)

Fermat's little theorem If p is prime, and $k \in \{1, 2, ..., p-1\}$, then $k^{p-1} \equiv 1 \pmod{n}$.

Proof that D is the inverse of E Dijk proves Euler's theorem and uses it to prove Fermat's little theorem in [DL]. He then uses Fermat's little theorem in the following proof.

$$D(E(m)) = (m^e)^d \bmod n = m^{ed} \bmod n$$
$$E(D(m)) = (m^d)^e \bmod n = m^{ed} \bmod n$$

by Equations 3 and 4.

$$\exists r \in \mathbb{Z} \cdot ed = 1 + r(p-1)(q-1)$$

by Equation 2. Hence,

$$m^{ed} = m^{1+r(p-1)(q-1)} = m \cdot m^{r(p-1)(q-1)}$$

Suppose $m \not\equiv 0 \pmod{p}$, then by Fermat's little theorem,

$$m^{p-1} \equiv 1 \pmod{p}$$
$$m^{ed} = m \cdot m^{r(p-1)(q-1)} = m \cdot (m^{p-1})^{r(q-1)} \equiv m \cdot 1 \equiv m \pmod{p}$$

Now suppose $m \equiv 0 \pmod{p}$, then

$$m^{ed} \equiv 0 \equiv m \pmod{p}$$

Therefore,

$$\forall m \in \mathbb{Z}_n \cdot m^{ed} \equiv m \pmod{p}$$

By symmetry,

$$\forall m \in \mathbb{Z}_n \cdot m^{ed} \equiv m \pmod{q}$$

This implies that

$$p \mid (m^{ed} - m) \land q \mid (m^{ed} - m)$$

but p and q are distinct primes, and $n = p \cdot q$, so

$$n \mid (m^{ed} - m)$$

Therefore,

$$m^{ed} \equiv m \pmod{n}$$

And since $m \in \mathbb{Z}_n$,

$$m = m^{ed} \bmod n$$

Hence, $D(E(m)) = E(D(m)) = m$, so the decryption of an encrypted message will work as expected. E is a bijective function on $\mathbb{Z}_n$, so it is a permutation as required in the public key cryptosystem scheme. The next section explains why it is also one-way.

Security

The RSA Problem and the Factoring Problem

The RSA problem (RSAP) is defined in [MKvOV18] as "given a positive integer n that is the product of two distinct odd primes p and q, a positive integer e such that $\gcd(e, (p-1)(q-1)) = 1$, and an integer c, find an integer m such that $m^e \equiv c \pmod{n}$." The given information in RSAP is the information exposed to the eavesdropper by the RSA algorithm, and solving RSAP is equivalent to finding the plaintext. Hence, the hardness of RSAP underlies the security of the RSA algorithm, as the security goal is to protect the confidentiality of m under the assumption that the private key is not compromised.

Rivest et al. conjecture that RSAP is computationally intractable [RSA78] (except a few special cases such as $c = 0$). It is closely associated with the well-studied factoring problem [Aum17], and both are widely believed to be equivalent [AM09, MKvOV18], but their equivalence has not been proven yet.

The factoring problem is defined in [Aum17] as "finding the prime numbers p and q given a large number $n = p \cdot q$". No polynomial-time algorithm has been found to solve the factoring problem, and the decision version of the problem is provably in **NP**, but it is believed to be easier than NP-complete problems although that has not been proven [GBGL08].

Security Level Based on the Factoring Problem

The factoring problem serves as the hardness assumption in the proof of RSA's security. [RSA78] proves that a few methods of breaking the RSA algorithm are at least as hard as the factoring problem by showing that the success of each attack implies easy factoring of n. If the factoring problem is hard, and the RSAP is equivalent to the factoring problem, then the RSA algorithm is hard to break, and the encryption procedure E is a one-way permutation.

Consequently, the security level of the RSA algorithm depends on the currently known fastest attack that solves the factoring problem. The asymptotically fastest method of factoring a number, n, is the general number field sieve, whose heuristic complexity is roughly $\exp(1.91 \times N^{1/3} \times (\log_2 N)^{2/3})$, where $N = \log_2 n$ [Aum17]. The National Institute of Standards and Technology (NIST) suggests that the modulus n in RSA be at least 3072 bits long for the algorithm to reach the security strength of 128 bits, comparable with that of the commonly used block cipher AES-128 [BBB+06].

Implementation and Performance

Choosing p and q

Supposing we want $n = p \cdot q$ to be 3072 bits, then p and q should be around 1536 bits in length. Their sizes should differ by a few digits so that both are sufficiently large and a simple attack by searching prime factors around $\sqrt{n}$ is unlikely to succeed [MKvOV18].

A number with 1536 binary digits has approximately $\log_{10} 2^{1536} \approx 462$ decimal digits. Using the method proposed in [RSA78], we can obtain a 462-digit prime number by generating 462-digit odd numbers randomly until we find a prime. From the prime number theorem [Jam03], there are approximately $x/\ln x$ primes less than or equal to x, so about $(x/2)/(x/\ln x) = (\ln x)/2$ random odd numbers need to be generated before a prime is found. For $x = 10^{462}$, $\ln x/2 \approx 532$. (Since we are only generating 462-digit prime numbers rather than any positive prime numbers less than x, this approximation assumes uniform distribution of prime numbers along the number axis.) The candidate numbers can be tested for primality using an efficient probablistic algorithm, such as the Solovay-Strassen primality test [SS77].

Choosing d and e

Either of d and e can be chosen first, and the other can be determined from the chosen value. Suppose we choose d first. The value we choose must be co-prime with $(p-1)(q-1)$ for Equation 2 to hold for some e, and this can be easily achieved by choosing any prime number greater than $\max(p, q)$ [RSA78], as any prime factors of $(p-1)$ or $(q-1)$ must be less than $\max(p, q)$. The

range of possible values of d must be large enough to render brute force attack infeasible [RSA78].

After choosing d, e can be computed using the extended Euclidean algorithm [Aum17]. Given inputs x and y, the extended Euclidean algorithm finds s and t such that $s \cdot x + t \cdot y = \gcd(x, y)$. Since $(p-1)(q-1)$ and d are relatively prime, we can use the algorithm to find integers s and e such that $s \cdot (p-1)(q-1) + e \cdot d = 1$. This implies that $e \cdot d \equiv 1 \pmod{((p-1)(q-1))}$, as required.

This algorithm takes $O(\log n)$ time to find e. Alternatively, it is common to fix e at $2^{16} + 1 = 65537$, which allows fast encryption with the square-and-multiply method due to its low Hamming weight. Moreover, the proportion of messages that encrypt to itself is negligible when e is small [MKvOV18].

Exponentiation

Square-and-Multiply

The encryption and the decryption in RSA are exponentiation operations modulo n. One family of fast exponentiation algorithms is square-and-multiply [Aum17], which computes $x^y \bmod n$ using $O(\log y)$ multiplication-like operations on numbers with the same size as n.

The Chinese Remaider Theorem

The Chinese remainder theorem is also commonly used to further accelerate exponentiation in the decryption procedure. It cannot be used in the encryption procedure as the secret factors are needed. To find $x^d \bmod n$ using the theorem, where $n = p \cdot q$, we have

$$x^d \bmod n = ((x^d \bmod p) \cdot q \cdot (q^{-1} \bmod p) + (x^d \bmod q) \cdot p \cdot (p^{-1} \bmod q)) \bmod n \tag{5}$$

This allows exponentiation under the much smaller moduli p and q, so the numbers that we need to square and multiply are smaller. The multiplicative inverse of p modulo q and that of q modulo p exist because $\gcd(p, q) = 1$ for distinct primes p and q, and they can be computed using the extended Euclidean algorithm. Moreover, $q \cdot (q^{-1} \bmod p)$ and $p \cdot (p^{-1} \bmod q)$ can be pre-computed to save more time [Aum17].

Application

[RSA78] aims to incorporate two properties of the paper mail system into an electronic mail system through the RSA algorithm. The first property is that "messages are private", which is a criterion for the security of a cipher. The second property is that "messages can be signed", which implies the use of digital

signatures. Both properties assume that the attacker has not compromised the private key.

Despite the algorithms that speed up the exponentiation in RSA, RSA is still slower than symmetric-key ciphers like the Advanced Encryption Standard (AES). Hence, RSA is typically used to transmit the symmetric keys of other ciphers and other small data like signatures [MKvOV18]. However, direct use of the RSA algorithm for these purposes is likely to fail the respective security goals, and extra features like randomness need to be added to make the entire algorithm more secure. The rest of this section describes some reasons for the insecurity and outlines stronger algorithms based on RSA as explained in [Aum17].

Public-Key Encryption

Textbook RSA Encryption

A secure encryption algorithm needs to produce ciphertexts with the properties of indistinguishability and non-malleability. The textbook RSA encryption simply takes the plaintext, m, and gives the ciphertext, $c = E(m)$. This algorithm is deterministic, as the same plaintext will give the same ciphertext every time it is encrypted with the same encryption key.

To test for indistinguishability, suppose the attacker picks two plaintexts and receives a ciphertext that corresponds to either of the two with probability $1/2$. The ciphertext is indistinguishable from a random string of bits if the attacker cannot tell with probability better than $1/2$ which plaintext was encrypted. Therefore, indistinguishability is impossible in the chosen-plaintext attackers (CPA) model, where the attacker can encrypt the plaintexts and compare the output with the given ciphertext.

Moreover, the naive encryption algorithm also yields malleable ciphertext. Suppose $E(m_1) = c_1$, and $E(m_2) = c_2$. Without knowing m_1 or m_2, the attacker can compute $c_3 = c_1 \cdot c_2 \bmod n = m_1^e \cdot m_2^e \bmod n = (m_1 \cdot m_2)^e \bmod n$, and know the relationship that $m_3 = D(c_3) = m_1 \cdot m_2 \bmod n$, thereby creating a valid ciphertext and deducing some information about its corresponding plaintext.

Optimal Asymmetric Encryption Padding

The Optimal Asymmetric Encryption Padding (OAEP) combined with RSA gives a stronger cipher, RSA-OAEP. The padding scheme uses a pseudorandom number generator (PRNG) to generate a seed, pads the plaintext according to some fixed algorithm, and mixes them using two unkeyed hash functions. The result, together with some intermediate values, are catenated to give the bit string P which will be encrypted by the RSA algorithm to give the final ciphertext S. After decrypting S, one can recover the seed and the plaintext

using the information in P. Without knowing the seed, the attacker is unable to tell which of the two plaintexts was encrypted in the test for indistinguishability. Since two plaintexts are almost certainly encrypted with different seeds and secure hash functions should be non-linear, it is hard for the attacker to create a ciphertext from existing ciphertexts and deduce something about the corresponding plaintext. Ciphertexts resulting from RSA-OAEP are therefore indistinguishable and non-malleable.

Digital Signature

Textbook RSA Signatures

A secure digital signature needs to satisfy unforgeability, which means that an attacker cannot make up a valid message-tag pair without knowing the private key. In the textbook RSA signature scheme, the owner of the private key signs the message m by creating $c = D(m)$, and everyone can verify the signature by computing $E(c)$ and checking that $E(c) = m$. However, this scheme allows trivial forgeries. The message-tag pairs $(0, 0)$, $(1, 1)$ and $(n-1, n-1)$ are always valid, independent of the private key.

This signature scheme is also weak against the blinding attack. Suppose the attacker wants the owner of the private key d to sign a message m unknowingly. The attacker may use some value k that is co-prime with n to create another message $m \cdot k^e \bmod n$, which the private key owner will sign consciously. Having obtained the signature $s = m^d \cdot k^{ed} \bmod n = m^d \cdot k \bmod n$, the attacker can compute $m^d \bmod n = s \cdot k^{-1} \bmod n$, which is the signature of m.

Probablistic Signature Scheme

The Probablistic Signature Scheme (PSS), like OAEP, also uses a PRNG and two unkeyed hash functions. It computes the hash value of the message m using the first hash function, pads this hash value with some '0's and a number R generated by the PRNG, and further mixes the value with extra padding and R using both hash functions. The output P is eventually passed into the RSA decryption procedure to generate the final signature S. To check the validity of the signature, one can compute P_1 by following the steps of PSS before the use of RSA, and $P_2 = E(S)$. The signature is verified if $P_1 = P_2$. (If the signature is valid, R can be recovered from $E(S)$ and used to compute P_1.)

Full Domain Hash Signatures

The Full Domain Hash (FDH) signature scheme simply computes the hash value of a message under a hash function, and signs that value using the RSA decryption procedure. It is almost as secure as PSS, but weaker against attacks like the Bellcore attack which target its deterministic nature.

Attacks

Forward Search Attack

If the message space is small, the attacker can find the message m corresponding to a ciphertext c by trying to encrypt possible messages in the message space until the ciphertext equals c. To avoid such attacks, the message can be salted with padding as in OAEP and PSS, so a brute-force search in the message space becomes infeasible [MKvOV18].

Common Modulus Attack

If two entities Alice and Bob have the same n for their own procedures, with key pairs (e_A, d_A) and (e_B, d_B), both of them will know p and q. (Even if Alice does not know p and q directly, she can compute them from n and (e_A, d_A), and so can Bob from his key pair [Aum17].) They can then easily compute the other entity's private key from the corresponding public key. Hence, it is crucial for different entities to have different n values [MKvOV18].

Timing Attack

The square-and-multiply algorithm makes it possible to recover the secret exponent d using a timing attack. The time taken by the algorithm is heavily dependent on each bit of d, as a branch to multiplication takes place exactly when the bit examined in that iteration is 1. By observing the timing, or the power consumption in a hardware implementation of the algorithm, the attacker may deduce d [Aum17].

Bellcore Attack

The Bellcore attack works on deterministic RSA implementations which use the Chinese remainder theorem. The attacker uses fault injection, forcing part of the algorithm to misbehave, and uses the faulty result to deduce the factors of n. For example, suppose the calculation of $x^d \bmod p$ in Equation 5 is faulty. For convenience, let

$$\begin{aligned}
y_p &:= x^d \bmod p \\
y_q &:= x^d \bmod q \\
y &:= x^d \bmod n = (y_p \cdot q \cdot (q^{-1} \bmod p) + y_q \cdot p \cdot (p^{-1} \bmod q)) \bmod n \\
y'_p &:= \text{the faulty result of } x^d \bmod p \\
y' &:= (y'_p \cdot q \cdot (q^{-1} \bmod p) + y_q \cdot p \cdot (p^{-1} \bmod q)) \bmod n
\end{aligned}$$

Then

$$y - y' = (y_p - y'_p) \cdot q \cdot (q^{-1} \bmod p)$$

So q is a factor of $y - y'$. y and y' are accessible to the attacker as the output of the decryption procedure in the chosen-ciphertext attacker (CCA) model. Hence, the attacker can find $q = \gcd(n, y - y')$, thereby factoring n [Aum17].

Solving the Factoring Problem with a Quantum Computer

If quantum computing becomes a reality, the RSA algorithm will no longer be secure. The factoring problem takes $O(2^N)$ time when solved by classical computers, where $N = \log_2 n$ is the size of the modulus n. Shor's algorithm which works on quantum computers solves the factoring problem in $O(n^2(\log n)(\log\log n))$ time, which is an exponential speed-up. If the hardness assumption underlying RSA is a problem that can be solved efficiently, then RSA can be broken as efficiently. Nevertheless, whether commercial-scale quantum computers can ever be manufactured is still a matter of debate [Aum17].

Conclusion

The RSA algorithm has stood the test of time since its publication, and cryptosystems based on it have proven reliable when implemented properly. Over the years, secure and more efficient public-key cryptography algorithms like the elliptic curve cryptography have emerged [Aum17] and may gradually take the lead, but the RSA algorithm remains a milestone in the history of cryptography.

References

[AM09] Divesh Aggarwal and Ueli Maurer. Breaking rsa generically is equivalent to factoring. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 36–53. Springer Berlin Heidelberg, 2009.

[Aum17] Jean-Philippe Aumasson. *Serious Cryptography*. William Pollock, 1st edition, 2017.

[BBB+06] Elaine Barker, William Barker, William Burr, William Polk, Miles Smid, et al. *Recommendation for key management: Part 1: General*. National Institute of Standards and Technology, Technology Administration, 2006.

[DH76] W Diffie and M Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DL] Marten van Dijk and Tom Leighton. 6.042j mathematics for computer science. fall 2010. https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2010/video-lectures/lecture-5-number-theory-ii/.

[GBGL08] Timothy Gowers, June Barrow-Green, and Imre Leader. *The Princeton companion to mathematics [electronic resource]*. Princeton University Press, 2008.

[HO00] Cordelia Hall and John O'Donnell. *Discrete mathematics using a computer*. Springer, 2000.

[Jam03] G. J. O Jameson. *The prime number theorem [electronic resource]*. London Mathematical Society student texts ; 53. 2003.

[MKvOV18] Alfred J. Menezes, Jonathan Katz, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography [electronic resource]*. Taylor & Francis Group, first edition, 2018.

[RSA78] R Rivest, A Shamir, and L Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[SS77] R. Solovay and V. Strassen. A fast monte-carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977.