Two's Complement, Fermat's Little Theorem and RSA Encryption

Shlomi Steinberg

February 2, 2021

1 Representation of Negative Integers: Two's Complement

Computers represent number in binary (base-2). Under binary, each digit is called a *bit* and can take two possible values: 0 or 1. Computers have fixed bit counts for representations of data (in the form of *registers*) and are capable of working with fixed data size. E.g., 32-bit processors can add and multiple 8/16/32-bit integers only. Unlike the binary representations that we did last week which can be arbitrary long (or short), we now assume that we have a fixed count of bits.

Two's complement is a binary representation scheme used to represent negative integers and essentially any computer uses this scheme. It isn't complicated, so we will dive right in: Assume that we work with 8-bit integers, we would like to write the negative integer -10 in binary using two's complement. First, we write out 10 in binary:

$$10 = (00001010)_2$$

Notice that we write all 8-bits. Now, we flip all the digits, add one to the result and that is the 8-bit two's complement representation of -10:

$$-10 = (11110110)_2$$

and that's two's complement.

The advantage of two's complement is that integer addition and subtraction requires no special treatment for negative integers. The most significant bit (the right-most bit) then always designates negative integers. When adding 1 to the largest possible positive integer $(01111111)_2 = 127$, it overflows to $(10000000)_2 = -128$.

When representing integers using two's complement scheme under any bit count, -1 will always have all binary bits set to 1. For example, under 8-bits:

$$-1 = (11111111)_2$$

when we add 1 to -1, the representation overflows to $(00000000)_2$, which is just 0 in binary, as desired.

2 Fermat's Little Theorem

Fermat's famous "Little Theorem" is stated as follows:

Theorem 2.1 (Fermat's Little Theorem). *Given p a prime integer, then for any integer a, $a^p - a$ is divisible by p. Equivalently,*

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

If a is not divisible by p, we also deduce:

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

Typical applications include calculating the modulo of very large powers of integers. As an example, let's calculate $2^{345} \pmod{11}$:

Example 2.1. 2 *is not divisible by* 11*, then by Fermat's Little Theorem,* $2^{11-1} \equiv 1 \pmod{11}$.

$$2^{345} = 2^{10 \cdot 34+5} = (2^{10})^{34} 2^5 \equiv 1^{34} 2^5 \equiv 32 \equiv 10 \pmod{11}$$

Euler's theorem We define *Euler's totient function* $\phi(n)$ to be the number of positive integers $1 \leq k \leq n$, such that $\gcd(k, n) = 1$.
For example: For any prime p, $\phi(p) = p - 1$. Consider the integer 10, it has 4 relatively prime integers less than it: $1, 3, 7, 9$; therefore $\phi(10) = 4$.

Euler's Theorem is then a generalization of Fermat's Little Theorem:

Theorem 2.2 (Euler's Theorem). *Given n and a relatively prime, then*

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

An interesting corollary immediately follows:

Corollary 2.1. *Given p, q distinct primes. Then for each integer a not divisible by p or q:*

$$a^{(p-1)(q-1)} \equiv 1 \pmod{pq}$$

3 Public-key Encryption: RSA

The most common encryption scheme is where one party encrypts a message using a *key* (passphrase or artefact) and a recipient who has access to the same key can decrypt and access the message. Public-key encryption is a strange development where the encryption and decryption are asymmetric one-way functions, but are intrinsically mathematically linked. The concept is very new: While forms of encryption and ciphers have appeared from ancient times, asymmetric public-private encryption scheme appears to have first originated in the British secret service in the early 70s, and then independently discovered by Diffie and Hellman in 1976, with a working system (RSA) presented a year later by Rivest, Shamir and Adleman. RSA is surprisingly simple and has stood the test of time remarkably well.

Imagine the following scenario: Alice and Bob want to exchange secret (private) information. However, they only have access to public communication channels and have not exchanged any private information (such as passphrases) a priori. How can Alice and Bob facilitate means of communication, secure from Eve who would like to intercept Alice's secrets?

RSA works as follows: A *public-key* generated by Alice is distributed freely and publicly. Alice also generates a corresponding *private-key* and stores it privately and securely. Bob can use the public-key to encrypt a message, but the public-key can not be used to decrypt the message! The encrypted message can be sent freely over public channels, and only Alice will be able to decrypt the message using her private-key. Similarly, Alice can *sign*, using her private-key, a message written by her and send it to Bob and Bob can use Alice's public-key to verify that the message has not been tempered with.

Keys generation We will use red colour for private information that must be secure from Eve, and blue for public information that we don't care if Eve gains knowledge of. Alice generates her key-pair by first selecting a pair of distinct prime numbers p and q. Alice proceeds by computing a modulus $N = p \cdot q$. Alice also chooses an encryption exponent e (*public key exponent*) that is relatively prime to $(p-1)(q-1)$. The encryption exponent is typically chosen to be a number of the form $2^n + 1$, for example $e = 65537$. Alice's public key is then: $\mathrm{pk} = (N, e)$.

Alice now generates her private-key: She computes an integer d (*private key exponent*) that satisfies

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

(i.e. the modular multiplicative inverse of e) for example, using the extended Euclidean algorithm (question 41-45, chapter 4.3 in the book). Alice private key is then: $\mathrm{sk} = (d, p, q)$. The private-key must be stored safely.

Before we discuss what those parts mean, let's see how they work.

Encryption and decryption Bob has access to Alice's public-key, $\mathrm{pk} = (N, e)$. Let $m < N$ be an integer representing part of a plaintext message. The encrypted *ciphertext*, c is then generated as follows

$$c = m^e \bmod N$$

The ciphertext can be distributed freely over public communication channels. Upon receiving the ciphertext, Alice can decrypt it using her private-key, $\mathrm{sk} = (d, p, q)$, as follows:

$$m = c^d \bmod N$$

Let's make sure that this actually works: Notice that $\phi(N) = \phi(pq) = (p-1)(q-1)$, and thus by Euler's Theorem 2.2 we know that $m^{\phi(N)} \equiv 1 \pmod{pq} \equiv 1 \pmod{N}$. Then,

$$\begin{aligned} c^d &= (m^e)^d \\ &= m^{ed} \\ &= m^{n(p-1)(q-1)+1} \\ &= \left(m^{\phi(N)}\right)^n m \\ &\equiv m \pmod{N} \end{aligned}$$

Note that raising m and c to large powers can be expensive! Fast repeated squaring is used for encryption part, which is why encryption is typically faster than decryption.

Example To make things clearer consider the following example: Suppose that Alice chooses randomly the primes $p = 29$, $q = 23$, and she selects $e = 3$ as her public exponent. Then, her public modulus becomes $N = pq = 667$ and the multiplicative inverse is $d = 411$. This defines a valid RSA cryptosystem, because e is relatively prime to $(p-1)(q-1)$.
Alice key-pair is then: The public-key $\text{pk} = (667, 3)$ and the private-key $\text{sk} = (411, 29, 23)$.

Alice publishes her public-key and Bob uses it to encrypt the top-secret message $m = 123$:

$$c = m^e \bmod N = 123^3 \bmod 667 = 604$$

Bob then send the ciphertext to Alice, who decrypts it using her private-key:

$$m = c^d \bmod N = 604^{411} \bmod 667 = 123$$

Why is this secure? Notice that no exchange of private (red) information was needed, all the transmitted data was public (blue) information. So, having only knowledge of public data, how can our adversary, Eve, get to the hidden message? The only thing that Eve needs is access to the private encryption exponent d. The most straightforward way to retrieve d is to factorize N into pq. But that is a HARD problem! If the selected primes p and q are very large (today, they are typically 2048-bit long, which is many hundreds of decimal digits), then it is impractical to factor those integers in a lifetime.

So the RSA problem is no more difficult then integer factorization. More importantly, is it less difficult? We do not know, in other words, we have not found an asymptotically faster method to solve for private encryption exponent. There is some evidence that it might be easier, however we don't know how much easier.

RSA remains a secure system against all conceivable threats. That will change over the next few decades with the advent of quantum computing. Specifically, integer factorization is one of the few practical problems that fall into the BQP class, i.e. problems that can be solved efficiently (in polynomial complexity) on a quantum computer, but a polynomial algorithm is not known for classical computers.

RSA is widely used. TLS (Transport Layer Security) uses RSA to negotiate a handshake between the client (you) and a server (e.g., any https website). They way this works is as follows: The server will serve a certificate, which contains an RSA public key. A key exchange will be performed, and then a symmetric encryption key will agreed upon for the session (because symmetric encryption is faster). To limit the dependence on trusted public keys, a method known as *Forward Secrecy* is used.

Finding large primes Finding large primes can be challenging. Surprisingly, there are deterministic primality test algorithms (AKS). The density of large primes smaller than 2^{2048} is also not too bad (the asymptotic density of primes is $\mathcal{O}\left(\frac{1}{\ln N}\right)$), roughly ≈ 0.001, i.e. a prime per a thousand integers on average. So, expected number of trials before finding a prime is about 1000. In practice, it is faster to use a probabilistic primality test which might discard valid primes.

However, it is crucial to remember that the selection of the primes remains random. The difficulty of factoring the integer N, which is what the entirety of our security is based upon, requires very large random primes. If an adversary, Eve, can guess something about our primes, e.g., a weakness in our random number generator resulting in predicable patterns or a specific structure of primes accepted by our primality test, then this can be exploited to reduce the effective search space, i.e. the entropy.

Other private-public key schemes RSA is the dominant but not the only system. Other systems, e.g., different elliptic curve systems, exist, however those systems are also broken by quantum computing.

Post-quantum cryptography: Symmetric schemes (like AES) remain secure. All RSA, Diffie-Hellman and elliptic curve schemes can be considered broken. Supersingular isogeny Diffie–Hellman key exchange (SIDH) is an example of a post-quantum private-public key scheme.