Contents

Chapter 1 Background

§ 1 Modular Arithmetic

§ 1.1 Modular Exponentiation

Theorem 1.1 (Fermat's Little Theorem)**.** Let p be a prime number, then

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

Furthermore, if a is coprime to p, an equivalent statement is that

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

Theorem 1.2 (Euler's Theorem)**.** A generalization of fermat's little theorem to arbitrary moduli. It states that for all a coprime to n the following holds

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

where ϕ is euler's totient function, defined as the amount of integers less than n that are coprime to n. Equivalently $\phi(n)$ is the order of the unit group of $\mathbb{Z}/n\mathbb{Z}$.

Proof. The residue classes of the integers coprime to n form a group under multiplication. The order of that group is $\phi(n)$. Let a be an element of this group and k be it's order. Lagrange's theorem implies that $k|\phi(n)$, so $\phi(n)$ is a multiple of k. We also know that $a^k \equiv 1 \pmod{n}$ per definition of the order. It follows that $a^{\phi(n)} \equiv a^{kM} \equiv (a^k)^M \equiv 1^M \equiv 1 \pmod{n}$. $m^{\lambda N} \equiv 1$ □

Definition 1.1 (Euler's Totient Function)**.** Euler's totient function, often also called euler's phi-function, or just ϕ of a positive natural number n is the amount of positive integers coprime to n less than n, or alternatively the order of the group of units of $\mathbb{Z}/n\mathbb{Z}$.

Definition 1.2 (Carmichael Function)**.** The Carmichael function $\lambda(n)$ is defined as the smallest positive integer m so that

$$a^m \equiv 1 \pmod{n}$$

for every a coprime to n. The value of the carmichael function is also called the exponent of the multiplicative group of integers modulo n.

Definition 1.3 (Quadratic Residue)**.** A number q is called a **quadratic residue** modulo n if it is congruent to a perfect square modulo n, meaning there exists x so that:

$$x^2 \equiv q \pmod{n}.$$

Otherwise, q is called a **quadratic nonresidue** modulo n.

Chapter 2 Discrete Logarithms

A discrete logarithm of an element b in a finite, cyclic group G, to a base $g \in G$ is a solution to:

$$g^x \equiv b.$$

There are many algorithms to calculate discrete logarithms, however if the order of g is large enough none of them suffice.

§ 2 Baby-step giant-step

This algorithm is good if the order of g is a prime, otherwise the pohlig-hellman algorithm is more efficient.

Algorithm 1 Shanks's Babystep-Giantstep Algorithm

Require: $g^x = h$
  $n \leftarrow 1 + \lfloor \sqrt{N} \rfloor$
  $h \leftarrow \{e, g, g^2, \dots, g^n\}$
  for do$j \in \{0, 1, \dots, n\}$
    $\text{table}[g^j] = j$
  end for
  $\gamma \leftarrow h$
  for all $i \in \{0, 1, \dots, n\}$ **do**
    if $\gamma \in \text{table}$ **then**
      return $in + \text{table}[\gamma]$
    else
      $\gamma \leftarrow \gamma \cdot g^{-n}$
    end if
  end for

Correctness

To prove that we always find a solution we need to show that there is always a match.

Proof. We rewrite x as follows:

$$x = nq + r, 0 \leq r < n$$

We can solve for q and since $n > \sqrt{N}$ we obtain:

$$q = \frac{x-r}{n} < \frac{N}{n} < \frac{N}{\sqrt{N}} = \sqrt{N} < n$$

Rewriting the discrete logarithm we get:

$$g^r = h \cdot g^{-nq}$$

And since $0 \leq r < n$, g^r is in the babysteps and because $q < n$, $-nq < -n^2$, so $h \cdot g^{-nq}$ is in the giant steps, therefore we always find a solution.

$\square$

Time/Space Complexity

Since we have $\mathcal{O}(1)$ lookup into the hashtable we have time complexity $\mathcal{O}(n) = \mathcal{O}(\sqrt{n})$. This consists of $2n$ modular exponentiations and n hash table inserts and lookups respectively.
Regarding space, we have a hash table storing $n+1$ elements, therefore we have $\mathcal{O}(\sqrt{n})$ space complexity.

§ 3 Pohlig-Hellman Algorithm

The idea is rather simple. We have a group G and want to find the discrete logarithm of a generator element $g \in G$, for some $h \in$, i.e. solve $g^x \equiv h \pmod{N}$, where $N = |G|$ is the order of G. We write $N = q_1^{e_1} \dots q_n^{e_n}$ where the q_i are the prime factors of N. We now look for solutions y_i for the sub-problems

$$g_i^y \equiv h_i \pmod{q_i^{e_i}}$$

where $g_i = g^{N/q_i^{e_i}}, h_i = h^{N/q_i^{e_i}}$. And find a solution x as a solution of the resulting system of congruence relations (by the chinese remainder theorem).

Solving the Sub-Problems

The naive approach to finding the y_is is of course to just apply some discrete logarithm algorithm, e.g. the babystep-giantstep algorithm described above. However, we can use a clever idea to speed up the process. To solve the discrete logarithm problem for an element of order q^e where q is a prime, we express an unknown solution as follows: $x = x_0 + x_1 q + x_2 q^2 + \dots + x_{e-1} q^{e-1}$. Therefore we have

$$\begin{aligned} h^{q^{e-1}} &\equiv (g^x)^{q^{e-1}} \\ &\equiv (g^{x_0 + x_1 q + x_2 q^2 + \dots + x_{e-1} q^{e-1}})^{q^{e-1}} \\ &\equiv (g^{x_0})^{q^{e-1}} (g^{q^e})^{x_1 + x_2 q + \dots + x_{e-1} q^{e-2}} \\ &\equiv (g^{q^{e-1}})^{x_0} \end{aligned}$$

Thus we can use the babystep-giantstep algorithm to solve $h^{q^{e-1}} \equiv (g^{q^{e-1}})^x \pmod{p}$. We get to x_1 by the same principle:

$$
\begin{aligned}
h^{q^{e-2}} &\equiv (g^{x_0+x_1q+x_2q^2+\cdots+x_{e-1}q^{e-1}})^{q^{e-2}} \\
&\equiv (g^{x_0+x_1q})^{q^{e-2}}(g^{q^e})^{x_2+\cdots+x_{e-1}q^{e-3}} \\
(hg^{-x_0})^{q^{e-2}} &\equiv (g^{q^{e-1}})^{x_1}
\end{aligned}
$$

And by following the same process we get the following definition for the i-th equation:

$$
(g^{q^{e-1}})^{x_i} \equiv (hg^{-x_0-x_1q-\cdots-x_{i-1}q^{i-1}})^{q^{e-i-1}}
$$

.
So, in order to solve the discrete logarithm for an element of order q^e we calculate all the x_i using the babystep-giantstep algorithm (or any other algorithm that is able to solve discrete log for elements with prime order), and combine the results to form the solution $x = x_0 + x_1q + \cdots + x_{e-1}q^{e-1}$.

§ 4 Index Calculus Algorithm

Chapter 3 Integer Factorization

Integer factorization is the problem of finding the prime factors (and their powers) of a given integer N. It is considered a hard problem and for large enough N there exist no efficient (pre-quantum) algorithms to solve it.

Chapter 4 RSA

Public-Key Cryptosytem based on the infeasibility of the integer factorization problem. Encrypt messages via modular exponentiation to a public key (consisting of an exponent and a modulus) and decrypt via a private key derived from the public key and the primes used to generate it.

§ 5 Key Generation

1. Choose (random) large primes p, q
2. Calculate $N = pq$ and $\lambda(N) = \operatorname{lcm}(\phi(p), \phi(q)) = \operatorname{lcm}(p-1, q-1)$
3. Choose a public exponent $3 \leq e < \lambda(N)$ with $gcd(e, \lambda(N)) = 1$
4. Calculate the private exponent $d \equiv e^{-1} \pmod{\lambda(N)}$

§ 5.1 Euler's Totient Function vs Carmichael Function

Originally RSA was formulated by using $\phi(N) = (p-1)(q-1)$ as the modulus for the inverse calculation, instead of $\lambda(N)$. Correctness for RSA is still guaranteed this way, but $gcd(e, \phi(N)) = 1$ is only a sufficient condition for a valid RSA public key, while $gcd(e, \lambda(N)) = 1$ is a necessary condition.

Proof. Clearly $\lambda(N)|\phi(N)$, as all orders in the multiplicative group divide ϕ and therefore if $\lambda(N)$ were greater than $\phi(N)$ it would not be the least common multiple. It follows that any tuple (d, e, N) generated using $\phi(N)$ is also valid for computations using the carmichael function, as

$$m^{ed} \equiv m^{1+k\phi(N)} \equiv m^{1+kM\lambda(N)} \equiv m^1 \pmod{N}.$$

On the other hand some tuple (d, e, N) generated using the carmichael function does not guarantee $gcd(e, \phi(N)) = 1$, as the following does not generally hold

$$ed \equiv 1 + k\lambda(N) \pmod{\phi N}$$

because $\lambda(N) \leq \phi(N)$ implies that $k\lambda(N)$ might not be a multiple of $\phi(N)$. □

§ 6 Encryption/Decryption

Alice wants to send m to Bob. Alice calculates

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

where (e, N) is Bob's public key and sends it to Bob. Bob receives c and decrypts it via:

$$c^d \equiv (m^e)^d \equiv m^{de} \equiv m \pmod N$$

This congruence relation works due to euler's theorem. Since we have $de \equiv 1 \pmod{\lambda(N)}$ it follows that $de = 1 + k\lambda(N)$, so $m^{de} \equiv m^{1+k\lambda(N)} \equiv m^1 m^{k\lambda(N)} \equiv m^1$.

§ 7 Attacks/Pitfalls

Don't implement RSA by yourself. There are endless mistakes that immediately compromise the security of the implementation. The following will list a bunch of these and explain how to exploit them.

§ 7.1 Factoring N from $\lambda(N)$ or $\phi(N)$

Assuming we somehow obtain the values for $\lambda(N)$ or $\phi(N)$ that were used when generating the RSA keys, we can obviously easily decrypt messages by calculating $d = e^{-1} \pmod{\lambda(N)}$, but can we also factor N directly? Yes! In the following we assume the standard practice of generating N as the product of two distinc primes p, q was used.

Knowing $\phi(N)$

We have $\phi(N) = (p-1)(q-1) = N - p - q + 1$. So $N - (p+q) + 1 = \phi(N)$ and $p + q = N - \phi(N) + 1$. Now look at $f(x) = (x-p)(x-q) = x^2 - (p+q)x + pq$, which has its roots at p and q. By substituting in $(p+q)$ we get an equation that only depends on N and $\phi(N)$:

$$x^2 - (N - \phi(N) + 1)x + pq$$

We obtain p and q by applying the quadratic formula:

$$p, q = \frac{N - \phi(N) + 1}{2} \pm \sqrt{\frac{(N - \phi(N) + 1)^2}{4} - 1}$$

Knowning $\lambda(N)$

§ 7.2 Factoring N from d

We can also factor N if we only have the private exponent d. The basis for the attack are square roots of 1 modulo N. If we have:

$$y^2 \equiv 1 \pmod N$$

$$y^2 - 1 \equiv 0 \pmod N$$

$$(y+1)(y-1) \equiv 0 \pmod N$$

because $p|N$ and $q|N$ and the zero product property holds, we know that:

$$y \equiv \pm 1 \pmod p \text{ and } y \equiv \pm 1 \pmod q.$$

If we find a solution to one of the following non-trivial system of equations:

$$y \equiv 1 \pmod p$$

$y \equiv -1 \pmod q$

(resp. the other non-trivial one), then we have $p|y-1$ and $p|N$, so we can factor N via $p = gcd(y-1, N)$.
To actually calculate the square roots we use the fact that if we have $a^b = c$ we can calculate $\sqrt{c}$ easily as $a^{\frac{b}{2}}$ if $2|b$. Note that the following therefore only works if $2|k$. This is certainly true if $\phi(N)$ was used during generation of the keys, because $\phi(N) = (p-1)(q-1)$ is obviously even, but I'm not sure if this is the case for the carmichael function as well. This leads us to the following algorithm:

Algorithm 2 Factoring N given d

Ensure: $k = 2^t r$, r odd
  $k \leftarrow ed - 1$
  loop
    $t \leftarrow k$
    $x \leftarrow \text{randrange}(2, N-1)$
    while t is even **do**
      $t \leftarrow t/2$
      $s \leftarrow \text{pow}(x, t, N)$
      if $x > 1 \wedge gcd(x-1, N) > 1$ **then**
        $p \leftarrow gcd(x-1, N)$
        return $(p, N/p)$
      end if
    end while
  end loop

There exist possible optimizations, e.g. using increasing primes instead of random choices for x[1].

§ 7.3 Broken RSA

This is based on a challenge on Cryptohack. In it, a prime p was used as the modulus and e was not coprime to $\lambda(p) = p - 1$. If $gcd(e, p-1) = 1$ it is extremely easy to decrypt the ciphertext, as $d \equiv e^{-1} \pmod p$ is easy to calculate. But as this was not the case, e is not invertible in $\mathbb{Z}/(p-1)\mathbb{Z}$. The trick is to divide out the common divisors of e and $\lambda(n)$ and try out all posible values for m, which are combinations of some arbitrary solution and e-th roots of unity.
Basically if we define $s = p_1 \dots p_k$ as the product of the shared prime factors of e and $\phi(n)$, the values $a^{\frac{\phi(n)}{s}} \pmod n$ for arbitrary a are all e-th roots of unity, since

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

because $\frac{e}{s}$ is an integer, as all factors of s are also factors of e by definition. It follows that all solutions of $x^e \equiv c \pmod n$ are of the form $x = x_0 * \zeta^i$ where ζ is a primitive e-th root of unity and x_0 is some initial solution. We obtain an initial solution easily via $x_0 \equiv c^d \pmod n$ with $d \equiv e^{-1} \pmod{\frac{\phi(n)}{s}}$. Clearly there are e e-th roots of unity and the fundamental theorem of algebra tells us that $x^e - c \equiv 0$ has (at most) e solutions, therefore all possible solutions are of this form.

[1]See this post for more info `https://crypto.stackexchange.com/questions/62482/algorithm-to-factorize-n-given-n-e-d`

§ 7.4 Small Private Exponent

There are two algorithms here, wiener's attack and boneh's attack.

Wiener's Attack

(The following assumes the ϕ-function was used when generating the keys)
If the private exponent is small enough, there is an efficient algorithm to recover d from the public key (e, N), called Wiener's attack. We know that $ed = k\phi(n) + 1$ and since $\phi(n) = pq - p - q + 1$ and $pq \gg p, q$ we can approximate $\phi(n) \approx n$. Our goal is to get to an approximation of the private key based on public information. We can get such a result by dividing $ed = k\phi(n) + 1$ by $d\phi(n)$:

$$\begin{aligned}\frac{e}{\phi(n)} &= \frac{k\phi(n)+1}{d\phi(n)} \\ &= \frac{k}{d} + \frac{1}{d\phi(n)} \\ &\approx \frac{k}{d}.\end{aligned}$$

Theorem 7.1 (Wiener's Theorem)**.** Let $n = pq$ with $q < p < 2q$. Let $d < \frac{1}{3}n^{\frac{1}{4}}$. Given (e, n) with $ed = 1 \pmod{\phi(n)}$, Mallory can efficiently recover d.

Proof. We first note that $n - \phi(n) < 3\sqrt{n}$, because $n - \phi(n) = p + q - 1 < p + q \leq 3q \leq 3\sqrt{n}$. And by using our approximation $\phi(n) \approx n$, we get $\frac{e}{n} \approx \frac{k}{d}$. We now calculate the error of this approximation:

$$\begin{aligned}\left|\frac{e}{n} - \frac{k}{d}\right| &= \left|\frac{ed - k\phi(n) - kn + k\phi(n)}{nd}\right| \\ &= \left|\frac{1 - k(n - \phi(n))}{nd}\right| \leq \left|\frac{3k\sqrt{n}}{nd}\right| = \frac{3k}{d\sqrt{n}}\end{aligned}$$

Now, $k\phi(n) = ed - 1 < ed$. From $e < \phi(n)$ we can deduce:

$$k < \frac{k\phi(n)}{e} < d < \frac{1}{3}N^{\frac{1}{4}}$$

Plugging this in (using $2d < 3d < n^{\frac{1}{4}} \implies \frac{1}{2d} > \frac{1}{n^{\frac{1}{4}}}$):

$$\left|\frac{e}{n} - \frac{k}{d}\right| \leq \frac{n^{\frac{1}{4}}}{d\sqrt{n}} = \frac{1}{dn^{\frac{1}{4}}} < \frac{1}{2d^2}.$$

Now, apparently if $\left|x - \frac{a}{b}\right| < \frac{1}{2b^2}$, then $\frac{a}{b}$ will be a convergent in the continued fraction expansion of x and the amount of convergents between x and $\frac{a}{b}$ is closely bounded by $log_2 n$. Therefore, we recover d in linear time (linear in the bitlength of n). $\square$

We obtain the same approximation if we use $\lambda(n)$, there's just an extra $gcd(p-1, q-1)$ in the equations that eventually disappears.