## 2.4 RSA and Number Theory

*"We know more than we did before. Let's use that."*

- Cypher

**Learning Outcomes**

Upon following these notes and the corresponding lecture, students will be able to

- explain how the RSA algorithm works, and describe why it is difficult for classical algorithms to break.
- apply properties about integers mod $N$.
- analyze the period finding algorithm and its correctness.

We've seen an example of private key cryptography when talking about quantum money, but here we will be interested in **public key cryptography**. In such a scheme, there exists what is called a public key, which anyone can easily know and uses to encrypt a message. On the other hand, each receiving party will have their own private key, which is required to efficiently decrypt a message.

In this section, we will take a look at the RSA protocol, which is one of the most commonly used public key cryptosystems today.

**Question 48.** Suppose the public keys are $e = 5$ and $M = 26$. If we encrypt the message "B" which we will represent using the integer 2, what is the cypher text?

**Question 49.** Show that $d = 17$ is a valid private key to decrypt the message.

More generally, the following is the description of the algorithm.

1. Choose two prime numbers $p$ and $q$.
2. Multiply them to form $M$.
3. Compute the "Euler function" of $M$, $\phi(M)$.

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26$$

4. Choose an encryption key $e$ such that
   - $1 < e < \phi(M)$.
   - $\gcd(e, M) = 1$ and $\gcd(e, (p-1)(q-1)) = 1$.
5. Choose $d$ such that

$$de(\text{mod } \phi(N)) = 1. \tag{71}$$

- Public keys are $e$ and $M$.
- Private keys are $p$, $q$, $d$.

If you know $p, q, e$, you can use Euclid's algorithm to efficiently compute $d$.

To encrypt a message, we would use the following protocol. Let $m$ be the plaintext message. The cypher text $c$ is computed by

$$c = m^e \mod M. \tag{72}$$

To decrypt the message, we simply perform

$$m = c^d \mod M. \tag{73}$$

If an attacker intercepted the cypher text $c$, it would be practically impossible to decode it unless they knew what $d$ was.

The computer science community believes that finding the plaintext $m$ from the cyphertext $c$ requires solving an exponentially hard problem. Knowing whether this is true or not has large implications for the security of information, leading to lots of interest in the complexity of the factoring problem.

As usual, let's take a look at how difficult it would be to factor numbers classically. Note that when we analyze the hardness of factoring, we are interested in the number of digits used to represent $M$, which is $w = O(\log M)$. So an efficient algorithm would mean an algorithm that runs in polynomial time with respect to $w = O(\log M)$.

The trivial algorithm would simply try every number $j \in [1, \sqrt{M}]$ and check if $j$ divides $M$. This would require $O(2^w)$ iterations.

Other known classical algorithms include

- Quadratic Field Sieve: $O\left(2^{c\cdot\sqrt{w}}\right)$
- Number Field Sieve: $O\left(2^{m^{1/3}}\right)$

Though still exponential, these improvements were enough to require RSA schemes to go up from 512- to 768-bit encryption schemes to the sizes we see today.

In 1994, Peter Shor developed an algorithm to solve factoring on a quantum computer using just poly($\log M$) gates. Note that this is not even the number of queries, it is the exact circuit size.

Going beyond RSA, another popular public-key cryptosystem is called Diffie-Hellman, which requires solving the discrete log problem (also believed to be difficult for classical computers). Shor's algorithm for factoring is also able to solve discrete log.

Shor's algorithm critically uses period finding, using a reduction from period finding to factoring. In other words, he shows how an efficient algorithm for period finding can be used to solve factoring.

We know that the quantum algorithm for period finding only uses $O(1)$ queries to $f$. We would like to see if the unitary representing the query can be efficiently implemented. This is possible to analyze because we are interested in a particular function when factoring, namely searching for the period of the function

$$f_x(s) := x^s \mod M \tag{74}$$

where $\gcd(x, M) = 1$ and $M = p \cdot q$. If we can find an efficient classical algorithm to compute $x^s \mod M$, we will know what the classical circuit looks like, then convert it into a reversible gate which will represent $U_f$.

In this section, we'll be proving some basic number theoretic facts related to the factoring problem. First, we need to confirm that $f_x(s)$ defined above is indeed periodic.

**Lemma 2.5.** Let $x$, $M$ be integers such that $\gcd(x, M) = 1$. Then

$$x^s \mod M \tag{75}$$

is periodic in $x$.

*Sketch.* If we tried computing the powers of $x$ mod $M$, since the function can have only $M$ distinct outcomes, if we take more than $M$ powers we will eventually get a repeat. Let $a$ and $b$ be two powers where the equation evaluates to the same integer. We can express this mathematically as

$$\begin{aligned} x^a \mod M = x^b \mod M && (76)\\ x^b - x^a = k \cdot M && (77)\\ x^a \left(x^{b-a} - 1\right) = k \cdot M && (78) \end{aligned}$$

for some integer $k$. Since $\gcd(x, M) = 1$, $x$ and $M$ do not share any prime factors. Thus for the equality to hold, $M$ must evenly divide $x^{b-a} - 1$:

$$\begin{aligned} x^{b-a} - 1 = k'M && (79)\\ x^{b-a} = 1 + k'M && (80)\\ \Rightarrow x^{b-a} = 1 \mod M. && (81) \end{aligned}$$

□

Furthermore, we can show that the period of the function is the smallest positive integer $s$ such that $x^s = 1 \mod M$.

In practice, finding a non-trivial square root of 1 mod $M$ is sufficient for factoring.

**Lemma 2.6.** Given a composite number $N$ and an integer $x$ such that

$$x^2 = 1 \mod N \tag{82}$$

and

$$x \neq \pm 1 \mod N, \tag{83}$$

we can factor $N$.

*Proof.* By our assumption, we have that

$$x^2 - 1 = kN \tag{84}$$

$$\Rightarrow (x+1)(x-1) = kN. \tag{85}$$

Furthermore by our second assumption that $x \neq \pm 1 \mod N$, neither $x - 1$ nor $x + 1$ is a multiple of $N$. The product is some multiple of $N$, so what we conclude is that each of $(x-1)$ and $(x+1)$ have some of $N$'s prime factors. To find one of these, we simply compute

$$\gcd(x+1, N) \tag{86}$$

$$\gcd(x-1, N). \tag{87}$$

□

**Example 2.7.** Let $N = 15$. A number that satisfies the first condition is $x = 11$:

$$11^2 = 121 = 1 \mod 15. \tag{88}$$

We can also easily verify that $11 \neq \pm 1 \mod 15$. Now we compute the gcd of the pair of integers around 11 with 15 to recover the factors:

$$\gcd(12, 15) = 3 \tag{89}$$

$$\gcd(10, 15) = 5. \tag{90}$$

We have successfully recovered the factors 3 and 5.

Note that for an application like RSA, the number $N$ we are trying to factor will always be a composite of two primes, meaning that the gcd will be sufficient for finding these numbers.

**Question 50.** Find a non-trivial square root of 1 mod 20.

## 2.5 Shor's Algorithm

We now have all the required pieces to see the full algorithm for factoring by Peter Shor.

**Algorithm 3** Factoring

```
Pick $x$ at random from $\{2, \ldots, N-1\}$
if $\gcd(x, N) \neq 1$ then
    $\gcd(x, N)$ is a non-trivial factor of $N$ so we are done!
else
    Use quantum Period Finding algorithm (alg 2) to find smallest $s$ such that $x^s = 1 \mod N$.
    Call this variable $r$.
    if $r$ is odd then
        Start over..
    else if $x^{r/2} = \pm 1 \mod N$ then
        Start over..
    else
        $x^{r/2} \mod N$ is a non-trivial square root of 1 mod $N$ ($\gcd(x^{r/2}-1, N)$.
    end if
end if
```

How likely is it that our algorithm actually finishes? For any $N$ that is not a power of a prime, if $x$ is chosen at random from $\mathbb{Z}_N^*$ and $r$ is the smallest $s$ such that $x^s = 1 \mod N$, then with probability $\geq 3/8$,

- $r$ is even, and
- $x^{r/2} \neq \pm 1 \mod N$.