8.4 The RSA Cryptosystem

So far, we have studied symmetric-key cryptosystems to allow two parties to communicate securely with each other when they share a secret key. We have also studied how two parties can establish a shared secret key using the Diffie-Hellman key exchange algorithm.

One of the limitations of symmetric-key cryptosystems is that a shared secret key needs to be established for every pair of people who want to communicate. If there are n people who each want to communicate securely with each other, there are $\frac{n(n-1)}{2}$ keys needed:

- The first person needs $n-1$ secret keys to communicate with everyone else.
- The second person needs $n-2$ secret keys to communicate with everyone else besides the first person.
- The third person needs $n-3$ secret keys to communicate with everyone else besides the first two people.
- This pattern repeats, for a total of $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$.

In this section, we'll introduce a new form of cryptosystem called a **public-key cryptosystem**, for each each person has two keys: a private key known only to them, and a public key known to everyone. We'll see what how to encrypt and decrypt messages in these cryptosystems, how they reduce the number of keys needed for people to communicate, and learn about the most widely-used public-key cryptosystem today, the RSA cryptosystem.

Public-key cryptography

A **public-key cryptosystem** is one where each party in the communication generates a *pair* of keys: a *private* (or *secret*) key, known only to them, and a *public* key which is known to everyone. Let's start with some intuition. Suppose Bob wants to send Alice a message using a public-key cryptosystem. Since he knows Alice's public key, he uses that key to encrypt the message, and sends her the ciphertext. Then, Alice uses her *private key* to decrypt the ciphertext.[1] Similarly, if Alice wants to send a message to Bob, she uses Bob's public key to encrypt the message, and Bob uses his private key to decrypt it.

[1] Recall that in a symmetric-key cryptosystem, messages are encrypted and decrypted with the same key–hence, the symmetry. Public-key cryptosystems are a form of *asymmetric* cryptosystem, since different keys are used for encryption and decryption.

More formally, we define a **secure public-key cryptosystem** as a system with the following parts:

- A set $\mathcal{P}$ of possible original messages, called **plaintext** messages. (E.g., a set of strings)

- A set $\mathcal{C}$ of possible encrypted messages, called **ciphertext** messages. (E.g., another set of strings)

- A set $\mathcal{K}_1$ of possible public keys and a set $\mathcal{K}_2$ of possible private keys.

- A subset $\mathcal{K} \subseteq \mathcal{K}_1 \times \mathcal{K}_2$ of possible **public-private key pairs**. Note that we use $\subseteq$ and not $=$ because not every public key can be paired with every private key.

- Two functions $Encrypt : \mathcal{K}_1 \times \mathcal{P} \to \mathcal{C}$ and $Decrypt : \mathcal{K}_2 \times \mathcal{C} \to \mathcal{P}$ that satisfy the following two properties:

    - (*correctness*) For all $(k_1, k_2) \in \mathcal{K}$ and $m \in \mathcal{P}$, $Decrypt(k_2, Encrypt(k_1, m)) = m$. (That is, if you encrypt and then decrypt the same message with a public-private key pair, you get back the original message.)
    - (*security*) For all $(k_1, k_2) \in \mathcal{K}$ and $m \in \mathcal{P}$, if an eavesdropper only knows the values of the public key k_1 and the ciphertext $c = Encrypt(k_1, m)$ but does not know k_2, it is computationally infeasible to find the plaintext message m.

The RSA cryptosystem

The Diffie-Hellman key exchange algorithm we studied in the last section worked by relying on the hardness of the *discrete logarithm problem*. This allowed Alice and Bob to communicate their numbers $g^a \mathbin{\%} p$ and $g^b \mathbin{\%} p publicly, without anyone being able to find the "secret" a and b.

The **Rivest-Shamir-Adleman (RSA) cryptosystem** works with numbers as well, and relies on the surprising hardness of factoring large integers. You could write a small Python program to answer this question quite quickly, but that was only a number with five digits. What about the number $1,455,980,635,647,702,351,701$, with 22 digits? In practice, RSA relies on the hardness of factoring integers with *hundreds* of digits!

Let's see how RSA works.

Phase 1: Key generation

Each person in a public-key cryptosystem must first generate a public-private key pair before they can communicate with anyone else. (Think about this as choosing a valid key-pair from the set $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2$.) For RSA, we'll put ourselves in Alice's shoes and see what she must do to to generate a key pair.

1. First, Alice picks two distinct prime numbers p and q.

2. Next, Alice computes the product $n = pq$.

3. Then, Alice chooses an integer $e \in \{2, 3, \dots, \varphi(n) - 1\}$ such that $\gcd(e, \varphi(n)) = 1$.[2]

4. Finally, Alice chooses an integer $d \in \{2, 3, \dots, \varphi(n) - 1\}$ that is the modular inverse of e modulo $\varphi(n)$.

    - That is, $de \equiv 1 \pmod{\varphi(n)}$.

[2] Remember from 7.5 Modular Exponentiation and Order that $\varphi(n)$ is the number of positive integers $< n$ that are coprime to n.

That's it! Alice's *private key* is the tuple (p, q, d), and her public key is the tuple (n, e). Alice shares her public key with the world, but she never tells her private key to anyone.

Phase 2: Message encryption

Now suppose that Bob wants to send Alice a plaintext message m. For now we'll treat the message as a number between 1 and $n-1$, and will discuss string messages later on in this section. Bob uses Alice's public key (n, e):

1. Bob computes the ciphertext $c = m^e \mathbin{\%} n$ and sends it to Alice.

Phase 3: Message decryption

Finally, Alice receives the ciphertext c. She uses her private key (p, q, d) to decrypt the message:

1. Alice computes $m' = c^d \mathbin{\%} n$.[3]

[3] Techincally, Alice can recompute n from the p and q of her private key. Another version of RSA is actually just to store n in the private key, or use the n from her public key (which Alice also has access to) and keep only d as the private key.

An example

Before moving on, let's see an example of a full use of the RSA cryptosystem in action. Alice first needs to generate a public and private key.

1. Alice chooses the prime numbers $p = 23$ and $q = 31$.
2. The product is $n = p \cdot q = 23 \cdot 31 = 713$.
3. Next, Alice needs to choose an e where $\gcd(e, \varphi(n)) = 1$. Alice calculates that $\varphi(713) = 660$, and chooses $e = 547$ to satisfy the constraints on e.
4. Finally, Alice calculates the modular inverse to find the last part of the private key ($d \cdot 547 \equiv 1 \pmod{660}$), so $d = 403$.

At the end of this phase:

- Alice's *private key* is $(p = 23, q = 31, d = 403)$.
- Alice's *public key* is $(n = 713, e = 547)$.

Now suppose Bob wants to send the number 42 to Alice. He computes the encrypted number to be $c = 42^e \mathbin{\%} n = 42^{547} \mathbin{\%} 713 = 106$ and sends it to Alice.

Alice receives the number 106 from Bob. She computes the decrypted number to be $m = 106^d \mathbin{\%} 713 = 106^{403} \mathbin{\%} 713 = 42$. Voila!

Proving the correctness of RSA

In the RSA cryptosystem, the encryption and decryption algorithms are very straightforward. The "interesting" part is in how the public-private key pair is generated to make the encryption and decryption work! In this section, we'll come to understand why the key generation involves the steps that it does by proving that the RSA algorithm works correctly, using all the number theory work we developed in the previous chapter.

Theorem. Let $(p, q, d) \in \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+$ be a private key and $(n, e) \in \mathbb{Z}^+ \times \mathbb{Z}^+$ its corresponding public key as generated by "RSA Phase 1". Let $m, c, m' \in \{1, \dots, n-1\}$ be the original plaintext message, ciphertext, and decrypted message, respectively, as described in the RSA encryption and decryption phases.

Then $m' = m$ (i.e., the decrypted message is the same as the original message).

Proof. Let $p, q, n, d, e, m, c, m' \in \mathbb{N}$ be defined as in the above definition of the RSA algorithm. We need to prove that $m' = m$.

NOTE: For the rest of this proof, we will introduce one additional assumption: that $\gcd(m, n) = 1$. (It is possible to prove this theorem without this assumption, but we will not do so here).

From the definition of m' in the decryption step, we know $m' \equiv c^d \pmod{n}$. From the definition of c in the encryption step, we know $c \equiv m^e \pmod{n}$. Putting these together, we have:

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

So we need to prove that $m^{ed} \equiv m \pmod{n}$. From Steps 3 and 4 of the RSA key generation phase, we know that $de \equiv 1 \pmod{\varphi(n)}$, i.e., there exists a $k \in \mathbb{Z}$ such that $de = k \cdot \varphi(n) + 1$.

We also know that since $\gcd(m, n) = 1$, by Euler's Theorem $m^{\varphi(n)} \equiv 1 \pmod{n}$.

Putting this all together, we have

$$\begin{aligned}
m' &\equiv m^{ed} && \pmod{n} \\
&\equiv m^{k \times \varphi(n) + 1} && \pmod{n} \\
&\equiv (m^{\varphi(n)})^k \times m && \pmod{n} \\
&\equiv 1^k \times m && \pmod{n} \quad \text{(by Euler's Theorem!)} \\
&\equiv m && \pmod{n}
\end{aligned}$$

So $m' \equiv m \pmod{n}$. Since we also know m and m' are between 1 and $n-1$, we can conclude that $m' = m$. ∎

The security of RSA

Now that we've established the correctness of the RSA cryptosystem, let's now discuss its security. As we did for the Diffie-Hellman key exchange, we'll put ourselves in the role of an eavesdropper who is trying to gain information about a secret message. Suppose we observe Bob sending an encrypted message c to Alice. In addition to the ciphertext, we also know Alice's public key (n, e).[4] What information can we hope to gain about Bob's original plaintext message?

[4] Remember that "public" means that *everyone* can see it—including possibly malicious users!

Approach 1: Reverse-engineering the message itself

First, we know from the RSA encryption phase that $c \equiv m^e \pmod{n}$, so if we know all three of c, e, and n, can we determine the value of m?

No! As we saw in in 8.3 Computing Shared Secret Keys, we don't have an efficient way of computing "e-th roots" in modular arithmetic—this is the *discrete logarithm problem*.

Approach 2: Determine Alice's private key from her public key

Another approach we could take is to attempt to discover Alice's private key. Recall that $de \equiv 1 \pmod{\varphi(n)}$. So d is the inverse of e modulo $\varphi(n)$, and we learned in the last chapter that we can compute modular inverses, so this should be easy, right?

Not so fast! We can compute the modular inverse of d modulo $\varphi(n)$ when we know both d and $\varphi(n)$, but right now we only know n, not $\varphi(n)$.

So how do we compute $\varphi(n)$? Well, we know that if $n = p \cdot q$ where p and q are distinct primes, then $\varphi(n) = (p-1)(q-1)$. But here is the problem: *it is not computationally feasible to factor n when it is extremely large*. This is our second "computationally hard" problem in computer science, the **Integer Factorization Problem**. Despite the best efforts of computer scientists and mathematicians for centuries, there is no known efficient general algorithm for factoring integers, and it is this fact that keeps the RSA private key (p, q, d) secure.