CS161 Notes
Topic: Asymmetric Cryptography
Lecturer: Dawn Song

The Shortcomings of Symmetric Cryptography

There is a very big disadvantage to using only symmetric encryption. For symmetric encryption to work in a distributed setting, lots of shared secrets need to be established ahead of time. More specifically, for n people, $O(n^2)$ keys are needed because each person needs to store $n-1$ keys (one for every other person). This is not scalable and is extremely hard to manage. So we need something better.

Ideally, we would like for Bob to have a public key and everyone who wants to send a message to Bob can look up his public key and encrypt a message such that only Bob can decrypt the message with his secret private key. This scheme requires that there be only n keys total (one for each person). Then, Alice will only need to store her private key locally and she can just look up the public key for Bob when she wishes to send a message to Bob.

Asymmetric Cryptography

Here is how a public-private key asymmetric algorithm works. Alice has a public key pk that she shares with everyone in the world (others can look up this public key somewhere or ask her for it). She also has a private key sk that she keeps *secret* from everyone.

Alice uses an asymmetric encryption function E to encrypt a message M with her public key pk to produce a ciphertext C as follows

$$C \leftarrow E(pk, M)$$

Then, Alice can send C to Bob over an untrusted communication channel. When Bob receives C, he can use the corresponding asymmetric decryption function D to decrypt the ciphertext C with the private key sk and recover the original message M

$$M \leftarrow D(sk, C)$$

Note that in order for this to be secure against passive attackers, the encryption should be *probabilistic* so that every time M is encrypted, a different ciphertext is generated. A non-probabilistic encryption function can be converted to a probabilistic one by applying random padding to the message while encrypting it. This prevents an active attacker from correlating identical messages. Furthermore, to be safe against active attackers, the message also needs to be authenticated. An active attacker can also reorder, remove, or duplicate messages, so for many applications, the messages must also include a counter that gets authenticated along with the message. The receiving party should then verify that the counter of the messages increases by one for each additional message.

Trapdoor Functions

The symmetric crypto concepts in the previous lecture (PRP and PRF) are insufficient to implement this kind of a public key system because PRPs and PRFs don't have the concept of public and private keys. Therefore, we need a new kind of primitive, trapdoor functions. A trapdoor permutation consists of the following constructs:

- A randomized key generation algorithm

$$G \rightarrow (pk, sk)$$

- An encryption function

$$F(pk, x) \rightarrow y$$

- A decryption function that is secure if $F(pk, \cdot)$ is "one way" without knowledge of sk. This means that computing $F^{-1}(sk, y)$ is computationally infeasible without knowing sk. For example, if it takes 2^{110} operations to recover a key, then this is considered computationally infeasible.

$$F^{-1}(sk, y) \rightarrow x$$

The RSA Trapdoor Permutation

Suppose that

$$n = 3 * 5 = 15$$

The integers from 1 to 15 that are relatively prime to 15 are denoted as Z_{15}^* and their values are

$$Z_{15}^* = \{ 1, 2, 4, 7, 8, 11, 13, 14\}$$

What is $2^x \ mod \ 15$ for different values of x? Notice that all powers of $2 \ mod \ 15$ fall into Z_{15}^*.

The size of Z_n^* is called Euler's totient function and is denoted as

$$\phi(n) = |Z_n^*|$$

For $n = 15$, we have

$$\phi(15) = |Z_{15}^*| = 8$$

If $n = p * q$ (p and q prime), then $\phi(n) = (p-1)(q-1)$.
Notice that $x^{\phi(n)} \ mod \ n$ is always 1.

Factoring n is a cryptographically hard problem. No one knows how to do if efficiently if p and q are large (e.g., 512 bits each). Also, if you know n, you still can't compute $\phi(n)$ efficiently.

Now we generate two numbers e, d such that $e * d = 1 \ mod \ \phi(n)$

- e and n are public
- d is private

We encrypt by computing $C = M^e \ mod \ n$

- M is the plaintext message.
- C is the ciphertext output.

Notice that

$$C^d = (X^e)^d = X^{e*d} = X^1 = X \quad (mod \ n)$$

This means that we can decrypt a ciphertext C by just computing C^d. Since d is private, only the owner of the private key can do this.

If you know p and q, then you can find a pair (e, d) . If you don't know p and q, but only know n, then it is not feasible to find d given e. Proof by contradiction: If Alice could get both e and d, then that would help Alice factor n to find p and q, but it is known that factoring is hard.

Secure Asymmetric Encryption

Alice wishes to send a message M to Bob and encrypt it with Bob's public key. Then Bob should be able to decrypt it with his private key. How can we do this?

We can perform a pure RSA encryption

- $C \leftarrow M^e \ mod \ n$

but this is not secure because:

- What if the message consists of zeroes? Then you always get zeroes and the adversary knows that.
- This is not probabilistic (randomized) encryption, meaning that the same message always encrypts to the same ciphertext. Therefore, an attacker can find out when the same message is sent even without knowing the message.
- If the attacker has external information (e.g., knows what one message is), then it might allow the attacker to determine when that same message gets sent again and the attacker can figure out some of the future messages.
- This is a malleable function. Specifically, if an attacker knows M^e, the attacker can construct $(k * M)^e$ for any k he chooses without knowing M. For example, the attacker can compute $E(M * 2)$ without knowing M.
    - This is susceptible to chosen plaintext attacks: Alice can pre-compute the ciphertexts for some set of possible messages and then match the ciphertexts to infer the messages.

In conclusion, RSA by itself is insecure. RSA padding techniques are needed to make it secure. Here is one way that we can build a secure public-key encryption/decryption algorithm from a trapdoor permutation like RSA.

Notation:

- $E'(k, M)$: Symmetric encryption of message M with key k.
- $D'(k, c)$: Symmetric decryption of ciphertext c with key k.
- $H(x)$: Cryptographic hash of x.
- $F(pk, x)$: The trapdoor permutation encryption of x with public key pk.
    - For example, pure RSA encryption
        - $C \leftarrow M^e \ mod \ n$
- $F^{-1}(sk, y)$: The trapdoor permutation decryption of y with private key sk.
    - For example, pure RSA decryption
        - $M \leftarrow C^d \ mod \ n$

To build a secure encryption $E(pk, M)$, do:

$x \leftarrow_R X$
$y \leftarrow F(pk, x)$
$k \ \leftarrow \ H(x)$
$c \ \leftarrow E'(k, m)$
Output (y, c)

To build a secure decryption $D\big(sk, (y, c)\big)$, do:

$x \ \leftarrow \ F^{-1}(sk, y)$
$k \ \leftarrow \ H(x)$
$M \ \leftarrow \ D'(k, c)$
Output M