1 Overview

1.1 Previously in ECS 153…

1.1.1 Digital Signatures Using RSA

If Alice wants to sign a message she is sending to Bob, she uses her private key:

$$\text{sign}(m) = c = h(m)^d \mod n$$

Bob then verifies the message using Alice’s public key:

$$\text{verify}(m, c) = \begin{cases} \text{true} & h(m) = c^e \mod n \\ \text{false} & \text{otherwise} \end{cases}$$

Note that the message is hashed. This prevents an attacker from getting an arbitrary message $m$ signed by getting a random message $m_r$ and $(m/m_r)$ signed, and then multiplying the two.

1.1.2 Ensuring Integrity of Public Keys

A trusted Certificate Authority (CA) signs the public key and identification (i.e., domain name), which together make up a certificate. The public key and identification must be signed together in order to prevent an attacker from mixing and matching. The CA’s public key is built into the browser and can be used to verify the certificate.

There are four types of certificate errors. From most to least malicious, they are:

1. $\text{verify}()$ returns false
2. Self-signed certificate
3. Domain name does not match
4. Certificate is expired

1.1.3 Certificate Trees

Signing every domain for a large institution (e.g., a university) is expensive and time-consuming, so servers can get a more expensive certificate that allows them to sign other certificates as “intermediate certificate authorities”. Browsers can follow the certification chain back to the root CA, whose public key is built-in.

1.2 Today’s Lecture

In this lecture we discuss the problem of how to exchange keys without the use of certificates or a trusted third party. We also explore methods of authentication, first focusing on the usage, storage, and transmission of passwords.
2 Main Section

We begin with the following problem: What if there is no trusted certificate authority? How do we engage in private communication with an arbitrary party? Specifically, how do we exchange keys? We solve this problem with Diffie-Hellman key agreement.

2.1 Diffie-Hellman Key Agreement

Solves the problem of "If two people have never met, how do they exchange keys?"

2.1.1 The Theory Behind It

A multiplicative group is defined as

\[ Z^n_\text{mod}_n = \{ a \in [1, n) | gcd(a, n) = 1 \} \]

If \( p \) is a prime number, then \( Z^*_p = [1, p - 1] \).

\( a \in Z^*_p \) is a generator of \( Z^*_p \) if for all \( b \in Z^*_p \) there exists an integer \( i \) such that \( a^i = b \mod p \)

2.1.2 The Protocol

1. Pick large prime \( p \), and generator \( a \in Z^*_p \)
2. Alice picks a random secret \( x \) such that \( 1 \leq x \leq p - 2 \)
   Alice sends Bob \( a^x \mod p \)
   Bob picks a random secret \( y \) such that \( 1 \leq y \leq p - 2 \)
   Bob sends Alice \( a^y \mod p \)
3. Alice and Bob’s shared key is now \( a^{xy} \mod p \)

This protocol is believed to be secure due to two assumptions.

2.1.3 CDH (Computational Diffie Hellman) Assumption

Given large prime \( p \), generator \( g \), \( g^a \mod p \) and \( g^b \mod p \), it’s difficult to compute \( g^{ab} \mod p \).

2.1.4 DDH (Decisional Diffie Hellman) Assumption

Given large prime \( p \), generator \( g \), \( g^a \mod p \), \( g^b \mod p \), and \( g^r \mod p \), it’s difficult to determine if \( g^{ab} = g^r \mod p \).

3 Cryptography in Practice

How would you authenticate yourself to a server?

Types of Authentication

- Based on something known, e.g. password
- Based on something inherent, e.g. biometrics
- Based on something possessed, e.g. RSA ID
3.1 Passwords

3.1.1 Password Cracking

There are several ways to crack a password.

- **Brute Force** Guessing every possible password. Usually too expensive for passwords of reasonable length.

- **Dictionary Attacks** Use a "dictionary" of common passwords (mostly actual words) as guesses. Can be avoided by choosing a strong password that would not be found in a dictionary.

- **Social Engineering** Tricking victims into giving the attackers their passwords. Commonly done through impersonating someone who has access to the passwords (e.g., phishing).

3.1.2 Password Hashing

What if a server is compromised? How do we prevent someone from retrieving and using passwords from a compromised server?

Passwords are vulnerable to replay attacks. Knowing this, what should be requested from the user? (Assuming the communication channel is secure.)

\[ A \rightarrow B : h(p) \text{ versus } A \rightarrow B : p \]

The server contains the hashed password, so if it requested the hashed password for authentication it could easily be replayed. This is why we send the unhashed password to the server and have it calculate the hash and authenticate it.

3.1.3 Password Salting

If there are \( n \) users and the bit length of a password is \(|p|\):

- To brute force all passwords: \( 2^{|p|} \)

- To brute force all passwords of a particular user: \( 2^{|p|} \)

- To brute force the password of any one user: \( \frac{2^{|p|}}{n} \)

This means that as the number of users increases, the more likely it is that an attack will be able to acquire a password to one of the accounts. To fix this, we salt passwords.

We concatenate the username \( u \) to the password and then hash it.

\[ h(u||p) \]

Now the number of tries needed is again \( 2^{|p|} \).

3.2 Hybrid Cryptography

Because public key cryptography is usually slower than symmetric key cryptography, public key cryptography is usually used to exchange symmetric keys. These are called session keys. Next time in ECS 153: the exciting story of how two parties exchange session keys!