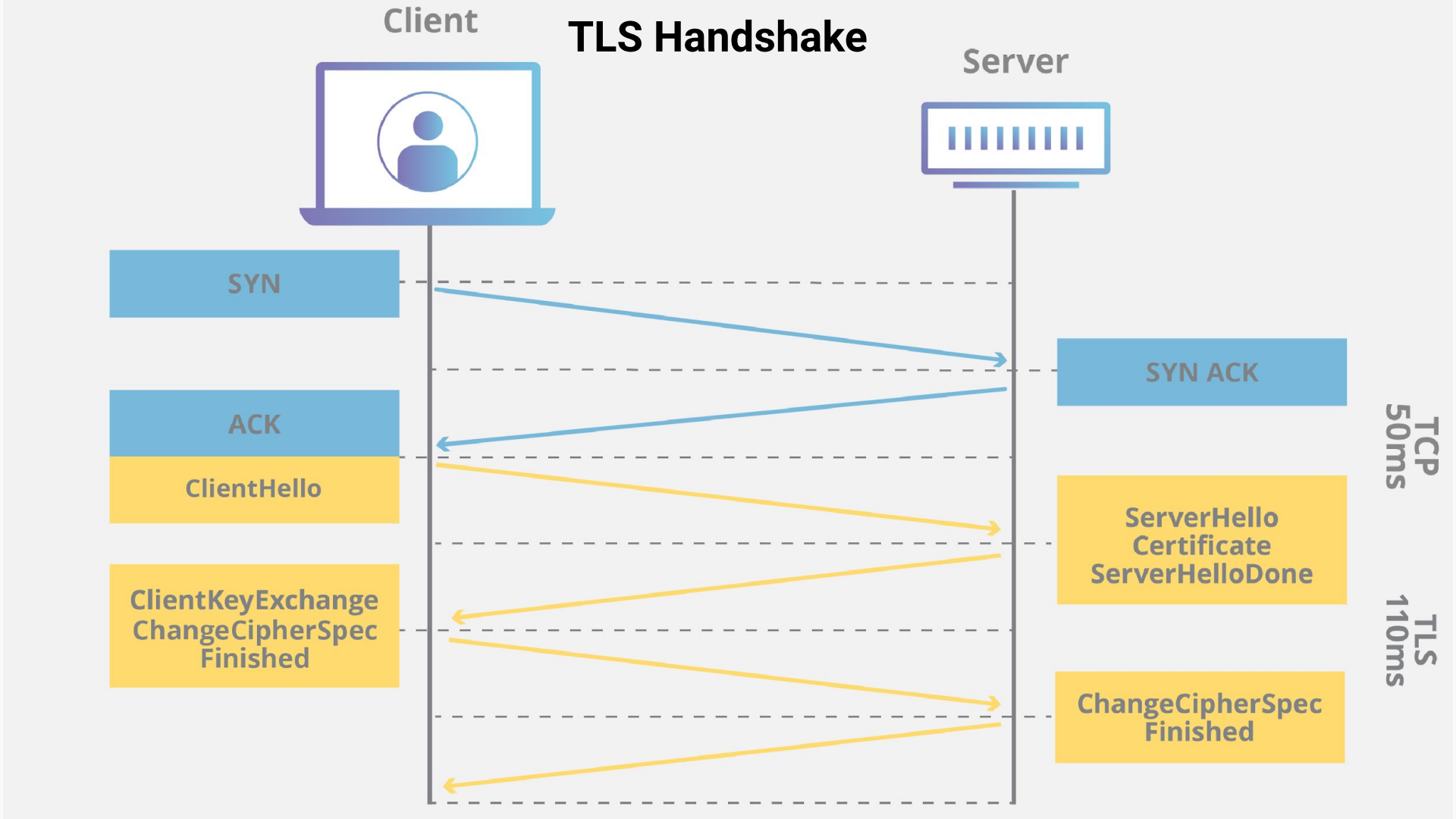
TLS Handshake and Certificate

HTTP vs HTTPS


Client
TLS Handshake
Server
SYN
SYN ACK
ACK
ClientHello
ServerHello
Certificate
ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished
TCP
50ms
TLS
110ms

Establish a TLS Connection

- ***Client Hello***
- Server Hello
- Server Certificate
- Server Key Exchange
- Server Hello Done
- Client Key Exchange
- Client Change Cipher Spec
- Client Handshake Finished
- Server Change Cipher Spec
- Server Handshake Finished

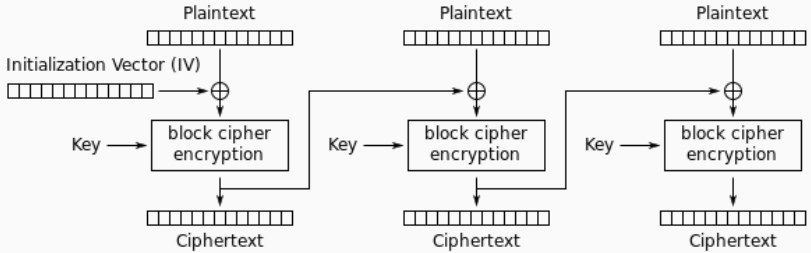
Establish a TLS Connection

Client Hello

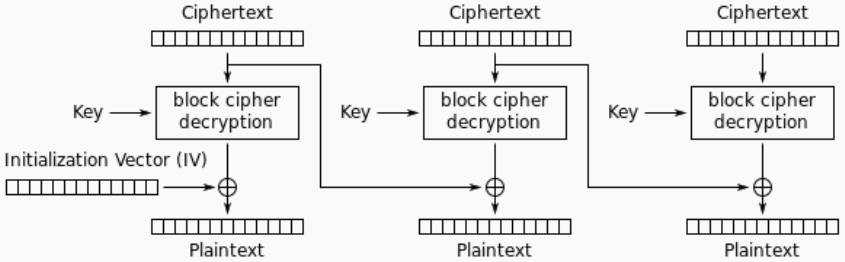
- Client version
- Client random
- Session ID
- Compression methods
- Cipher suites

CVE-2011-3389: BEAST (Browser Exploit Against SSL/TLS)

- Popular configurations of TLS use cipher block chaining (CBC) mode



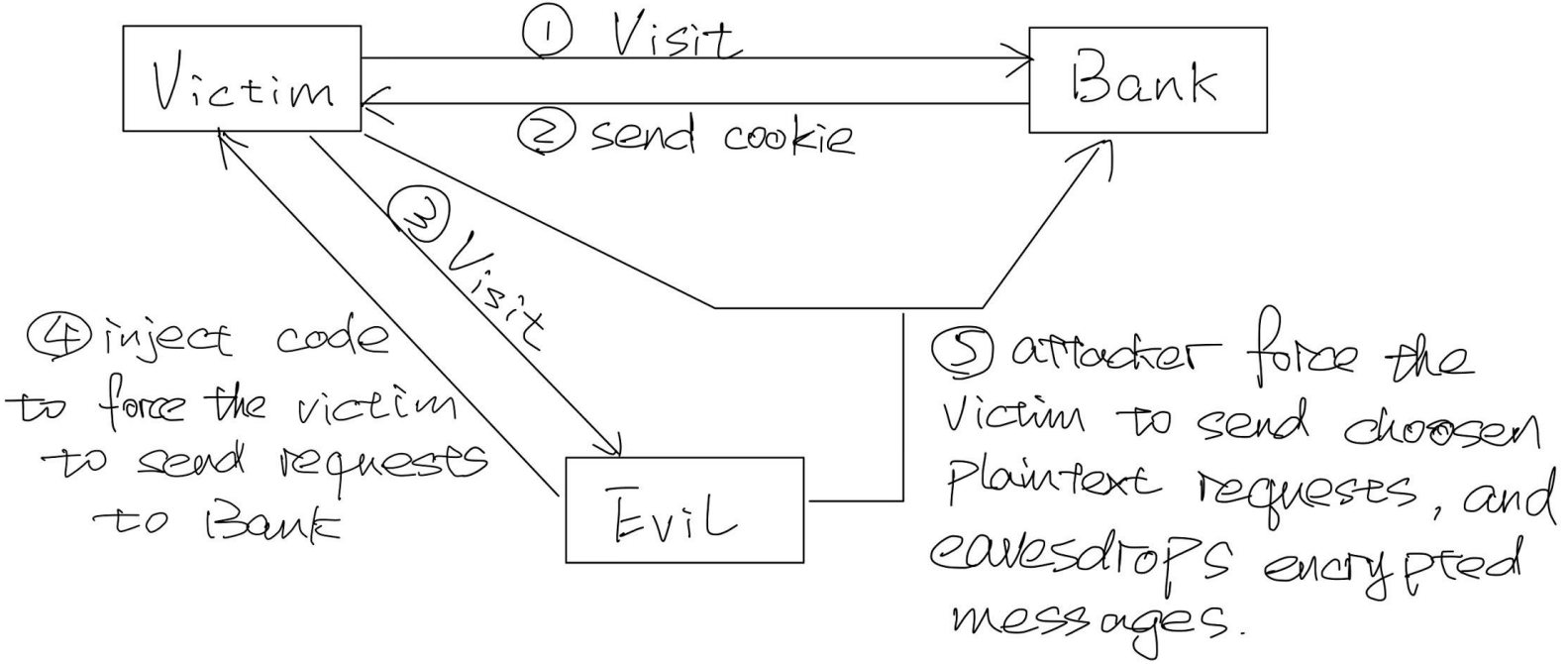
Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

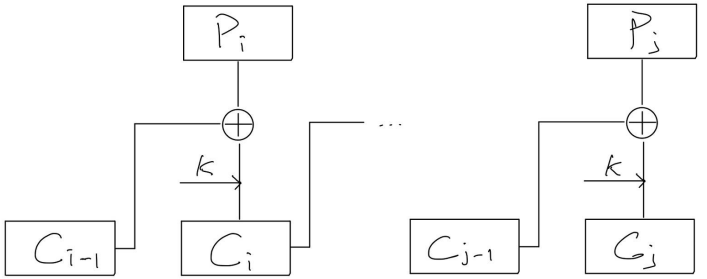
CVE-2011-3389: BEAST (Browser Exploit Against SSL/TLS)

- Attack Scenario



CVE-2011-3389: BEAST (Browser Exploit Against SSL/TLS)

- Attack



We want to find P_i, let X be our guess

$$\begin{cases} C_i = E(K, C_{i-1} \oplus P_i) \\ C_j = E(K, C_{j-1} \oplus P_j) \end{cases}$$

let $P_j = C_{i-1} \oplus C_{j-1} \oplus X$

$$C_j = E(K, C_{j-1} \oplus C_{i-1} \oplus C_{j-1} \oplus X)$$

$$C_j = E(K, C_{i-1} \oplus X)$$

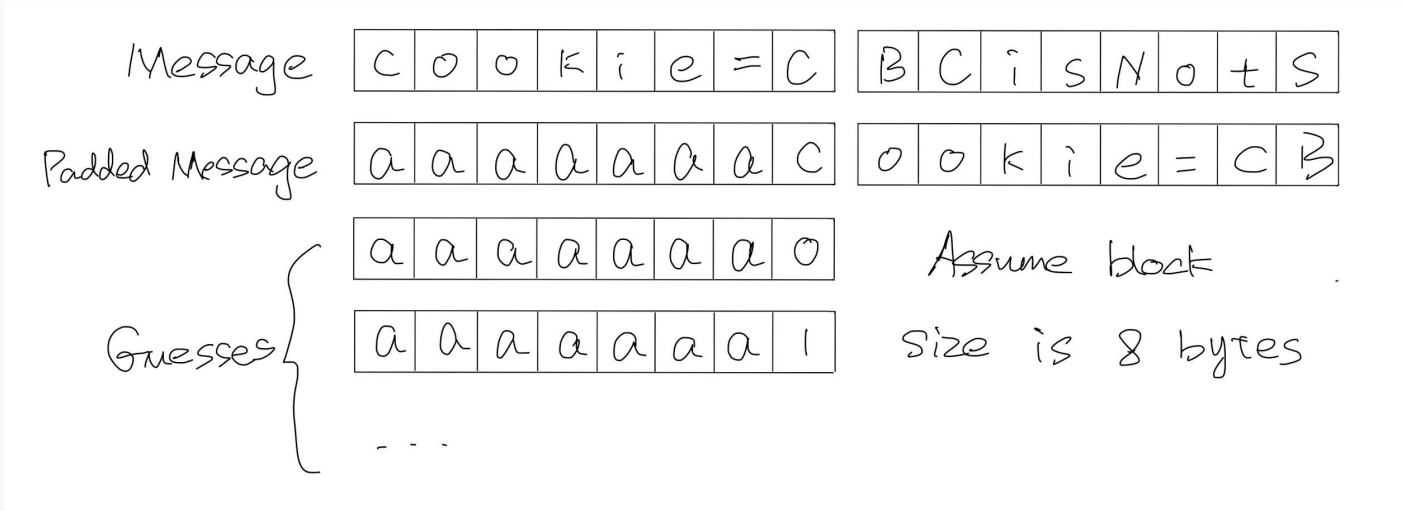
if $X = P_i$ (our guess is correct)

$$C_j = E(K, C_{i-1} \oplus X) = E(K, C_{i-1} \oplus P_i) = C_i$$

CVE-2011-3389: BEAST (Browser Exploit Against SSL/TLS)

- As you might notice, stealing a block of 16 bytes (256 bits) data requires 2^{256} guesses, which makes the attack nearly impossible to be carried out.
- However, by carefully padding the block with known data to the point where only the last byte of the block is unknown, we only need 2^8=256 guesses which is realistic for real world scenario.

- How can the attacker pad the message then?
- The attacker knows what a typical HTTP request looks like, he/she can control the request plain text by modifying the 'path' attribute in the HTTP request header, therefore control the padding.



CVE-2012-4929: CRIME (Compression Ratio Info-leak Made Easy)

- Common compression method replace repeated byte sequences with a pointer to the first instance of the sequence, thus reduce the message size

- The attacker will abuse this TLS compression method to hijack users' session by stealing their cookies
- Suppose victim is browsing his banking website and the website uses cookie to identify the session
- The attacker will host a malicious website which has multiple <img> tags in it, and all tags have path to the victim's banking website
- When victim visits attacker's malicious website, the <img> tag will be automatically loaded, thus trigger the browser to send requests to the banking website without user's notice
- Since TLS compress data from mixed source, the <img> path and victim's valid cookie will be compressed together
- The attacker then can sniff the packets and get the size of the requests that are sent
- By brute forcing each character, the attacker can steal the cookie from user eventually

Cipher suites example

Cipher suites are identified by strings.

A sample cipher suite string is: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

This string contains the following information:

- TLS is the protocol being used
- ECDHE is the key exchange algorithm (Elliptic Curve Diffie–Hellman)
- ECDSA is the authentication algorithm (Elliptic Curve Digital Signature Algorithm)
- AES_128_GCM is the data encryption algorithm (Advanced Encryption Standard 128 bit Galois/Counter Mode)
- SHA256 is the Message Authentication Code (MAC) algorithm (Secure Hash Algorithm 256 bit)

CVE-2020-0601: CurveBall Vulnerability

- A spoofing vulnerability exists in the way Windows CryptoAPI (Crypt32.dll) validates Elliptic Curve Cryptography (ECC) certificates.
- Elliptic Curve Cryptography Signature Algorithm (ECDSA) is a digital signature algorithm which uses Elliptic Curve Cryptography
- In ECDSA, we can use the private key k and generator G to compute public key $P_k = k \cdot G$
- It is easy to compute a generator given private key P_k and public key k
- Normally, the ECC certificate validation algorithm would need to check whether the certificate has a standardized generator
- However, Windows CryptoAPI wouldn't check for the generator if there is a valid cached certificate to be matched
- Therefore, we can craft a " valid certificate" (has the same public key as the real one) using a fake generator

Establish a TLS Connection

- Client Hello
- ***Server Hello***
- ***Server Certificate***
- ***Server Key Exchange***
- ***Server Hello Done***
- Client Key Exchange
- Client Change Cipher Spec
- Client Handshake Finished
- Server Change Cipher Spec
- Server Handshake Finished

Establish a TLS Connection

- Client Hello
- Server Hello
- Server Certificate
- Server Key Exchange
- Server Hello Done
- ***Client Key Exchange***
- Client Change Cipher Spec
- Client Handshake Finished
- Server Change Cipher Spec
- Server Handshake Finished

Client Key Exchange

- The **pre-master secret** is created by the client (the method of creation depends on the cipher suite) and then shared with the server.

- There are several key exchange algorithms: **RSA, DH**...
  - **RSA**: client encrypts randomly chosen premaster secret with the server's RSA public key
  - **DH**: client and server securely exchanging cryptographic keys as premaster secret

Diffie-Hellman

- Alice and Bob agree on modulus p = 23 and base g = 5 (which is a primitive root modulo 23)
- Alice chooses a secret integer a = 4, then sends Bob A = g^a mod p
  - A = g^a mod p = 5^4 mod 23 = 4
- Bob chooses a secret integer b = 3, theeen sends Alice B = g^b mod p
  - B = g^b mod p = 5^3 mod 23 = 10
- Alice computes S = B^a mod p
  - S = B^a mod p = 10^4 mod 23 =18
- Bob computes s = A^b mod p
  - S = A^b mod p = 4^3 mod 23 = 18
- Alice and Bob now share a secret (the number S = 18)
- Proof: A^b mod p = $(g^a \text{ mod } p)^b$ mod p = $(g^a)^b$ mod p
- $(g^a \text{ mod } p)^b$ mod p = $(g^b \text{ mod } p)^a$ mod p

Client Key Exchange

- The **pre-master secret** is created by the client (the method of creation depends on the cipher suite) and then shared with the server.

- There are several key exchange algorithms: **RSA, DH**...
  - **RSA**: client encrypts randomly chosen premaster secret with the server's RSA public key
  - **DH**: client and server securely exchanging cryptographic keys as premaster secret

- Then both client and server use the **premaster secre**t with **client_random** and **server_random** (sent in ClientHello and ServerHello messages) to generate a **master secret**.

- The **master secret** will be used to generate 4 session keys:
  - Client write key: encrypt client to server messages
  - Server write key: encrypt server to client messages
  - Client write MAC key: digitally sign client to server messages
  - Server write MAC key: digitally sign server to client messages

Establish a TLS Connection

- Client Hello
- Server Hello
- Server Certificate
- Server Key Exchange
- Server Hello Done
- Client Key Exchange
- ***Client Change Cipher Spec***
- ***Client Handshake Finished***
- ***Server Change Cipher Spec***
- ***Server Handshake Finished***

Establish a TLS Connection

Client Change Cipher Spec

- The Change Cipher Spec protocol is used to change the encryption method. Any data sent by the client from now on will be encrypted using the **symmetric shared key**.

Client Handshake Finished

- The last message of the handshake process from the client signifies that the handshake is finished. This is also the first encrypted message of the secure connection.

Server Change Cipher Spec

Server Handshake Finished

How to Obtain a Certificate

Step 1: provide domain name alongside with other information about your website

Step 2: generate a pair of public private key

Step 3: sign the information with the newly generated private key and send it to CA

Step 4: the CA will verify the **Certificate Signing Request**

Step 5: the CA will sign the certificate with its private key and send it back to the client

Step 6: the client add trusted certificate to the web server

What is Inside a Certificate

- Certificate
  - Version Number
  - Serial Number
  - Signature Algorithm ID
  - Issuer Name
  - Validity period
    - Not Before
    - Not After
  - Subject name
  - Subject Public Key Info
    - Public Key Algorithm
    - Subject Public Key
  - Issuer Unique Identifier (optional)
  - Subject Unique Identifier (optional)
  - Extensions (optional)
    - ...
- Certificate Signature Algorithm
- Certificate Signature