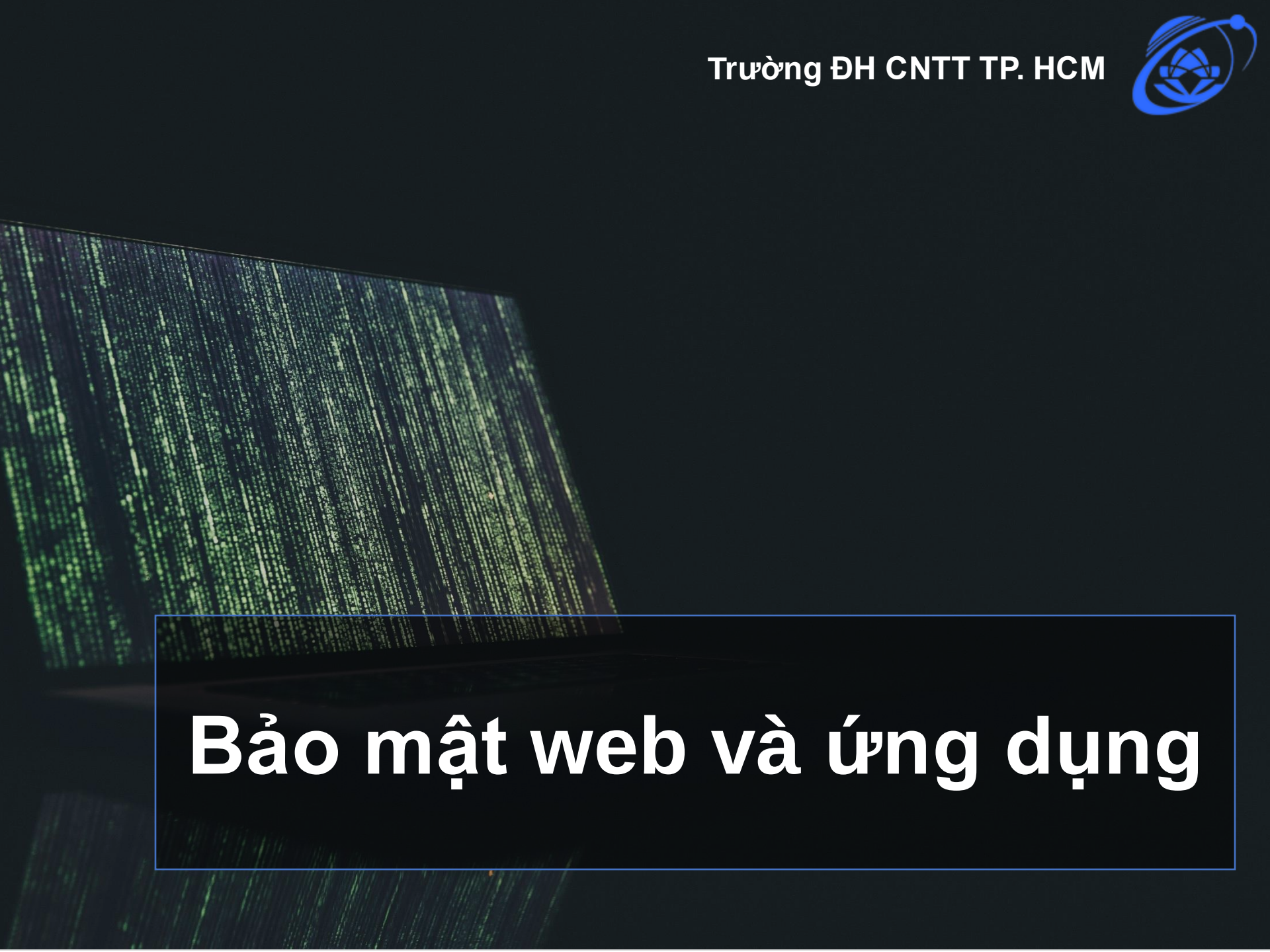
Trường ĐH CNTT TP. HCM



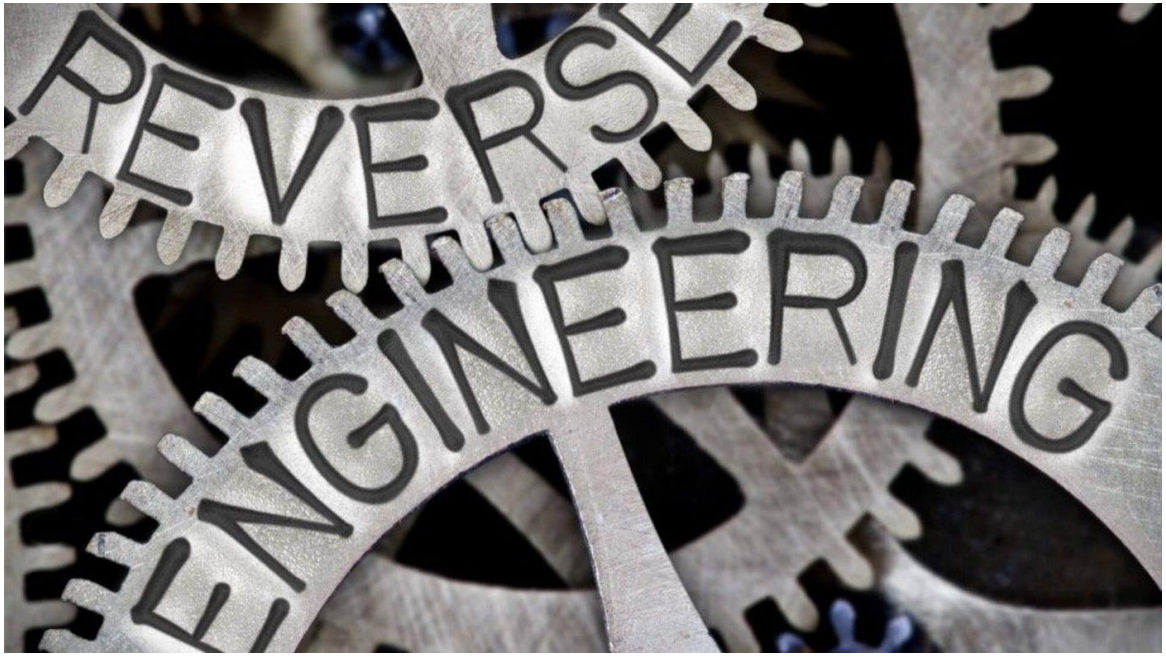
Bảo mật web và ứng dụng

Nội dung

- Bảo vệ dữ liệu lưu trữ (Stored Data)
- An toàn kết nối ứng dụng với Server

Credential Storage

- Làm thế nào để bảo vệ dữ liệu lưu trữ trong ứng dụng?
- Reverse Engineering?



Credential Storage

- Mã hóa đối xứng
  - AES, Blowfish, CAST5, RC4, 3DES
- Mã hóa bất đối xứng
  - Diffie-Hellman, RSA, ECDSA, ElGamal
- Kỹ thuật Hash
  - SHA-1, SHA-2, MD5, Tiger
- Password-based encryption (PBE)
- Key Derivation Function (KDF)

Credential Storage
PBE

- Android hỗ trợ thuật toán mã hóa (**Android password-based encryption – PBE)** sử dụng **SHA-256** và **AES**, để tính khóa mã hóa đối xứng từ password (hashing, mã hóa, thực hiện nhiều lượt chu kỳ mã hóa).
- Khi user cần đọc dữ liệu mã hóa → lấy key từ password để giải mã dữ liệu

```
String password = ...;

String PBE_ALGORITHM = "PBEWithSHA256And256BitAES-CBC-BC";
int NUM_OF_ITERATIONS = 1000;
int KEY_SIZE = 256;
```

1

```
byte[] salt = "abababababababababab".getBytes();

try
{
    PBEKeySpec pbeKeySpec = new PBEKeySpec(password.toCharArray(),
        salt, NUM_OF_ITERATIONS, KEY_SIZE);
    SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(PBE_ALGORITHM);
    SecretKey tempKey = keyFactory.generateSecret(pbeKeySpec);
    SecretKey secretKey = new SecretKeySpec(tempKey.getEncoded(), "AES");
}
catch (Exception exp)
{
    ...
```

2

PBE

- **Lưu ý:**
  - Salt dùng để nối thêm vào mã hash của password → tăng độ khó của việc tìm ra khóa
  - Để luôn có được key giống nhau từ password giống nhau, cần đảm bảo salt luôn cố định → Salt cần được cố định hardcode trong mã nguồn ứng dụng
  - Salt cần duy nhất cho mỗi thiết bị, được sinh ra lúc cài đặt ứng dụng

Credential Storage

Mã hóa không dữ liệu từ người dùng?

- Yêu cầu password cho mỗi lần đọc/ghi thông tin (mã hóa/giải mã)→ phiền toái cho người dùng.
- Lưu key hardcode trong mã nguồn → rủi ro reverse engineering ứng dụng.
- Giải quyết bằng cách sử dụng chiến lược Key Derivation lúc khởi động app, sau đó lưu cache key này lại → rủi ro reverse engineering ứng dụng.
- Mức độ bảo vệ dữ liệu phải PHÙ HỢP với mức độ rủi ro.

?? Lưu key trên key storage ở máy chủ (server)

Practical Cryptography

- VD: Sử dụng **Initialization Vector (IV)** kèm KDF để mã hóa/giải mã dữ liệu.
- Tìm hiểu đặc điểm của **Initialization Vector (IV)?**

```
String password = ...;

String PBE_ALGORITHM = "PBEWithSHA256And256BitAES-CBC-BC";
String CIPHER_ALGORITHM = "AES/CBC/PKCS5Padding";
int NUM_OF_ITERATIONS = 1000;
int KEY_SIZE = 256;

byte[] salt = "abababababababababababab".getBytes();
byte[] iv = "1234567890abcdef".getBytes();

String clearText = ...; // This is the value to be encrypted.
byte[] encryptedText;
byte[] decryptedText;
```

Practical Cryptography

```
try
{
    PBEKeySpec pbeKeySpec = new PBEKeySpec(password.toCharArray(),
        salt, NUM_OF_ITERATIONS, KEY_SIZE);
    SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(PBE_ALGORITHM);
    SecretKey tempKey = keyFactory.generateSecret(pbeKeySpec);
    SecretKey secretKey = new SecretKeySpec(tempKey.getEncoded(), "AES");

    IvParameterSpec ivSpec = new IvParameterSpec(iv);

    Cipher encCipher = Cipher.getInstance(CIPHER_ALGORITHM);
    encCipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);

    Cipher decCipher = Cipher.getInstance(CIPHER_ALGORITHM);
    decCipher.init(Cipher.DECRYPT_MODE, secretKey, ivSpec);

    encryptedText = encCipher.doFinal(clearText.getBytes());

    decryptedText = decCipher.doFinal(encryptedText);
    String sameAsClearText = new String(decryptedText);
}
catch (Exception e)
{
    ...
```

Practical Cryptography

- Tóm lại:
  - Các dữ liệu nhạy cảm, quan trọng cần được mã hóa
  - Khóa mã hóa **KHÔNG** nên lưu ở bất kỳ đâu, cũng như bất kỳ lúc nào

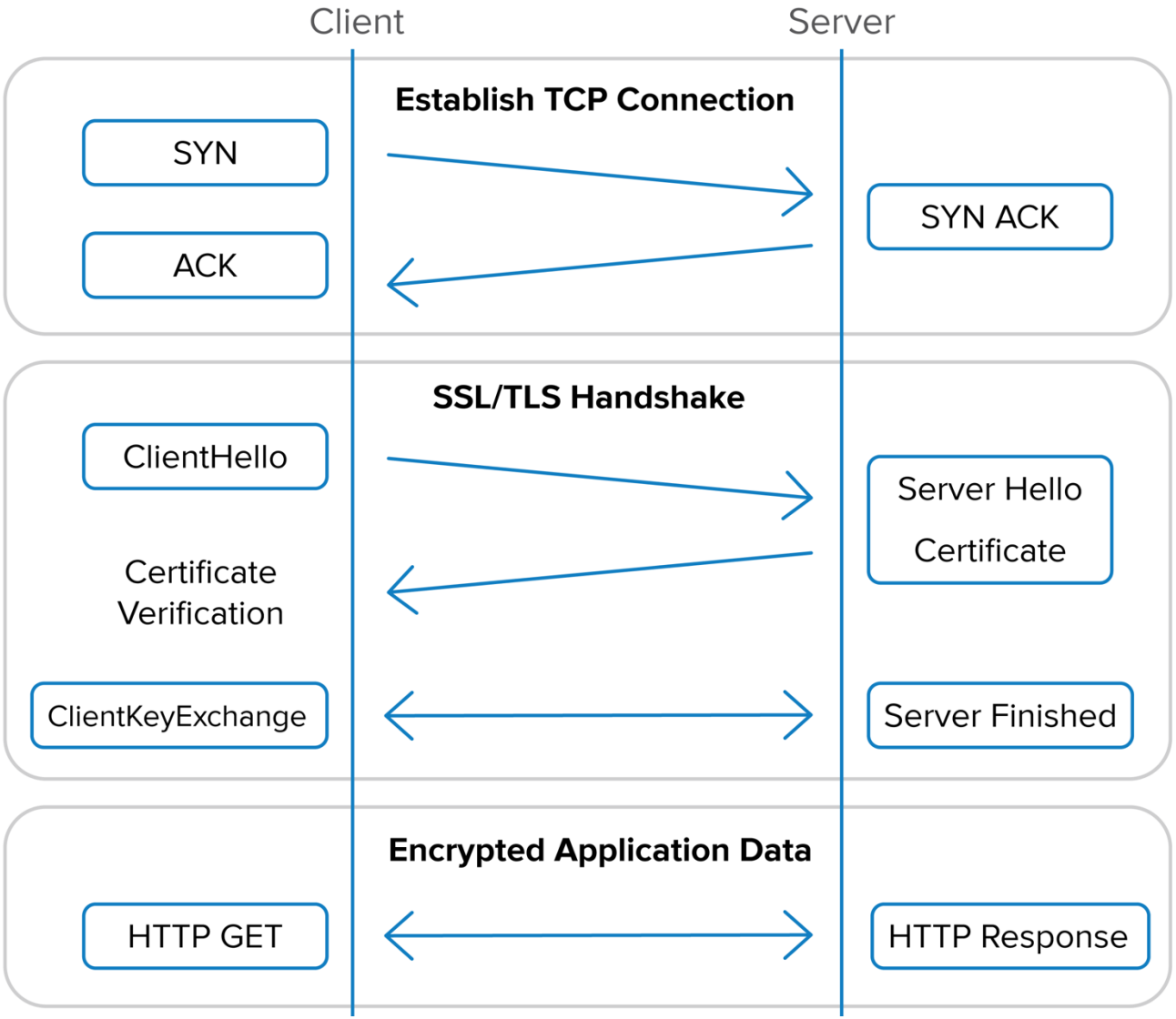
An toàn kết nối với Server

- **Tính bí mật** (confidentiality): ngăn chặn dữ liệu truyền nhận qua mạng bị đọc lén/ chỉnh sửa bởi các bên không liên quan.
- **Tính xác thực** (authentication): xác thực đối tượng máy chủ mà dữ liệu gửi tới, và dữ liệu nhận được có phải là đối tượng mong muốn hay không.

SSL/TLS

- Bảo vệ tính bí mật của dữ liệu truyền qua mạng
- Cho phép client/ứng dụng xác thực máy chủ mà nó đang kết nối, truyền nhận dữ liệu tới.

SSL/TLS


Client
Server
Establish TCP Connection
SYN
SYN ACK
ACK
SSL/TLS Handshake
ClientHello
Server Hello
Certificate
Certificate
Verification
ClientKeyExchange
Server Finished
Encrypted Application Data
HTTP GET
HTTP Response

SSL/TLS trong bảo vệ dữ liệu đến các server công cộng

Server công cộng

- Server nằm ngoài quyền quản lý mà người phát triển ứng dụng Android muốn kết nối tới.
- Các server này thường được cấu hình với các chứng chỉ từ những CA phổ biến.
- Mỗi phiên bản Android có chứa một danh sách các CA phổ biến mà nó tin tưởng.

Tạo kết nối HTTPS đến server công cộng

- **Sử dụng lớp HttpsURLConnection:** hiện thực SSL/TLS trên class chuẩn HttpURLConnection

```
URL url = new URL("https://clientaccess.example.com");
HttpsURLConnection urlConn = (HttpsURLConnection)url.openConnection();
```

Hoặc:

```
HttpsURLConnection urlConn = new HttpsURLConnection("https://
clientaccess.example.com");
```

- **VD:**

```
try
{
    URL url = new URL("https://clientaccess.example.com");
    HttpsURLConnection urlConn = (HttpsURLConnection)url.openConnection();
    urlConn.setDoOutput(true);
    OutputStream output = urlConn.getOutputStream();
    InputStream input = urlConn.getInputStream();
}
```

Tạo kết nối HTTPS đến server công cộng

Một số thông tin trong đối tượng

HttpsURLConnection:

- **getCipherSuite():** thuật toán mã hóa dùng trong kết nối
- **getServerCertificates():** trả về các chứng chỉ mà máy chủ cung cấp.

Xác thực máy chủ công cộng

- **Hostname verification**: so khớp hostname của server muốn kết nối tới với chứng chỉ của server đó

Việc xác thực được hiện thực với các lớp (class) sử dụng ***HostnameVerifier*** interface, trong Android có 1 số class có chức năng này:

- **AllowAllHostnameVerifier:** tắt tính năng hostname verification, chấp nhận tất cả chứng chỉ được tin cậy
- **StrictHostnameVerifier:** so sánh chứng chỉ dựa vào wildcard 1 cấp
  - VD: *.example.com (đây là wildcard)
    server1.example.com (match)
    server1.domain1.example.com (không match)
- **BrowserCompatHostnameVerifier:** so sánh chứng chỉ dựa vào hostname với wildcard nhiều cấp
  - VD: *.example.com (đây là wildcard)
    server1.domain1.example.com (match)

Xác thực máy chủ công cộng

- Ví dụ: Thực hiện tạo lớp xác thực **hostname** chứng chỉ với method **verify()**, khi đó:
  - URL sẽ được kiểm tra với thông tin đối tượng **SSLSession** của chứng chỉ.

```
HostnameVerifier customHV = new HostnameVerifier()
{
    public boolean verify(String urlHostname, SSLSession connSession)
    {
        String certificateHostname = connSession.getPeerHost();

        // compare urlHostname and certificateHostname here and
        // return true if the certificate should be accepted and
        // false if it should be rejected.
    }
};
```

Xác thực máy chủ công cộng

- Chỉ định HostnameVerifier cho các kết nối SSL/TLS trong tương lai bằng cách sử dụng phương thức ***setDefaultHostnameVerifier()*** trong lớp **HttpsURLConnection**

```
HostnameVerifier newHV = new StrictHostnameVerifier();
HttpsURLConnection.setDefaultHostnameVerifier(newHV);
```

Xác thực máy chủ công cộng

- VD: Kiểm tra thêm nhiều thông tin của chứng chỉ trước khi thiết lập kết nối an toàn (ngoài wildcard)

```
HostnameVerifier customHV = new HostnameVerifier()
        {
            public boolean verify(String urlHostname, SSLSession connSession)
            {
                boolean isCertOK = true;

                String certificateHostname = connSession.getPeerHost();
                // If we want to, can check to see the name of the host we connected to
                // and make a decision based on that information on if we trust it.
                try {
                    Certificate[] certs = connSession.getPeerCertificates();
                    X509Certificate caCert = (X509Certificate)(certs[certs.length - 1]);
                    String caName = caCert.getIssuerX500Principal().getName();

                    if ( caName == CA_WE_DO_NOT_WANT_TO_TRUST )
                    // Check to see if the certificate was issued by a CA that we
                    // choose to reject. If so, we'll reject it.
                    {
                        isCertOK = false;
                    }
                }
                catch (SSLPeerUnverifiedException e)
                // If we cannot verify the host, reject it.
                {
                    isCertOK = false;
                }

                return isCertOK;
            }
        };
```

SSL/TLS trong bảo vệ dữ liệu đến các server riêng tư

Server riêng tư

- Server cụ thể do chính nhà phát triển ứng dụng client quản lý.
- Được xây dựng để phục vụ cho ứng dụng phía client.

→ *Mô hình áp dụng với các server công cộng nằm ngoài quyền quản lý có thể không phù hợp*

Xác thực máy chủ riêng tư

- Có thể cấu hình SSL/TLS chỉ cho phép sử dụng một số chứng chỉ nhất định.
- Cho phép giới hạn các server mà ứng dụng có thể kết nối.
- Cho phép SSL/TLS tin tưởng các chứng chỉ không phải ký bởi các trusted CA, ví dụ self-signed certificate.

→ Có thể hiệu quả khi giao tiếp với server riêng tư (an toàn mà không cần mua các chứng chỉ từ CA thương mại)

- ***Ví dụ:***

Cấu hình cho ứng dụng Android tin tưởng chứng chỉ tự ký (self-signed) và từ chối tất cả chứng chỉ khác

Xác thực máy chủ riêng tư

- **B1**: Chuẩn bị chứng chỉ tự ký cho kết nối SSL an toàn giữa ứng dụng Android và máy chủ:
  - Có nhiều cách tạo chứng chỉ cho máy chủ web server
  - Dùng công cụ ***keytool*** trong Android

  VD: tạo chứng chỉ cho hostname *server.example.com* lưu trong keystore với định dạng JKS

```
keytool -genkey -dname "cn=server.example.com, ou=Development, o=example.com, c=US" -
alias selfsignedcert1 -keypass genericPassword -keystore certsjks -storepass
genericPassword -validity 365
```

- **B2**: lưu trữ khóa private ở web server

Xác thực máy chủ riêng tư

- **B3**: Tạo **BKS keystore** cho chứng chỉ tự ký để dùng trong ứng dụng Android.
  - Thực hiện: Cài đặt thư viện mã hóa Bouncy Castle vào Android
  - Sử dụng keytool để export chứng chỉ tự ký từ JKS keystore và import vào BKS keystore
  - VD: tạo BKS Keystore, chứa trong thư mục resource của ứng dụng (res/raw/selfsignedcertsbks)

```
keytool -export -alias selfsignedcert1 -keystore certsjks -storepass genericPassword -
keypass genericPassword -file cert.cer

keytool -import -file cert.cer -keypass genericPassword -keystore /home/user1/dev/
project1/res/raw/selfsignedcertsbks -storetype BKS -storepass genericPassword -
providerClass org.bouncycastle.jce.provider.BouncyCastleProvider -providerpath /home/
user1/lib/bcprov-jdk16-146.jar -alias selfsignedcert1
```

Xác thực máy chủ riêng tư

- **B4**: Cấu hình ứng dụng Android thiết lập kết nối SSL/TLS bằng cách sử dụng các đối tượng:
  - **KeyStore**: chứa các chứng chỉ và khóa bí mật
  - **TrustManager**: đối tượng chịu trách nhiệm quyết định các thông tin về chứng chỉ của những thực thể nào được tin cậy và chấp nhận trong kết nối SSL/TLS.
  - **TrustManagerFactory**: tạo đối tượng TrustManager
  - **SSLContext**: đối tượng chứa danh sách các thuộc tính cấu hình SSL (các khóa của client-side, các chứng chỉ server-side nào được chấp nhận – thông qua chỉ định TrustManger)
  - **SSLSocketFactory**: tạo socket SSL/TLS với các thuộc tính trong SSLContext

Xác thực máy chủ riêng tư

Sử dụng chứng chỉ tự ký đã thêm trong kết nối SSL/TLS của ứng dụng Android

- B4.1: Tạo KeyStore để truy cập keystore vừa thêm trong ***res/***

```
KeyStore selfsignedKeys = KeyStore.getInstance("BKS");
selfsignedKeys.load(context.getResources().openRawResource(R.raw.selfsignedcertsbks),
 "genericPassword".toCharArray());
```

- B4.2: Tạo TrustManagerFactory: cấu hình TrustManagerFactory với keystore để tạo TrustManager chỉ tin tưởng chứng chỉ trên

```
TrustManagerFactory trustMgr =
    TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
trustMgr.init(selfsignedKeys);
```

- B4.3: Tạo SSLContext để thiết lập kết nối SSL/TLS sử dụng chứng chỉ hostname đã tạo trước đó

```
SSLContext selfsignedSSLcontext = SSLContext.getInstance("TLS");
selfsignedSSLcontext.init(null, trustMgr.getTrustManagers(), new SecureRandom());
HttpsURLConnection.setDefaultSSLSocketFactory(
    selfsignedSSLcontext.getSocketFactory());
```

Xác thực máy chủ riêng tư

Sử dụng chứng chỉ tự ký đã thêm trong kết nối SSL/TLS của ứng dụng Android

- B4.4: Sử dụng cấu hình ở trên để thiết lập kết nối SSL/TLS cho ứng dụng với chứng chỉ đã tạo:

```
URL serverURL = new URL("https://server.example.com/endpointTest");
HttpsURLConnection serverConn = (HttpsURLConnection)serverURL.openConnection();
```

An toàn dữ liệu đầu vào

- SQL Injection
- Command injection

Kiểm tra dữ liệu đầu vào

- Kiểm tra tính hợp lệ của các dữ liệu đầu vào trước khi thực hiện xử lý.
- 2 hướng kiểm tra:
  - *Reject-Known-Bad*: reject hết những dữ liệu đã biết là xấu

  → yếu, do danh sách các dữ liệu đầu vào thế nào là xấu không thể xác định hết
  - *Accept-Known-Good*:
    - Data Type
    - Length
    - Numeric Range
    - Numeric Sign
    - Syntax/Grammar

Tài liệu tham khảo

- Chương 5-6-7 Sách **“Application Security for the Android Platform: Processes, Permissions, and Other”.** Jeff Six. 2011.

- Chương 5-6-7 sách **“Android Security Internals : An In-Depth Guide to Android's Security Architecture”**. Nikolay Elenkov. No Starch Press. 2015.

Bài tập

Xây dựng 1 ứng dụng Android sử dụng kết nối SSL/TLS giữa máy chủ và ứng dụng người dùng.

Bảo mật web và ứng dụng

Trường ĐH CNTT TP. HCM