**The Architectures Of Information Security**

***Resources from Microsoft Copilot***

Talking about information security, we talk about different kinds of information encryption and decryption methods. This article we will explore the custom or traditional ways of the architechture behind. For web browsing, not talking about ways of information encrytion, we will start talking about the architechture behind of how web browsing handles information security. We will first discuss what “**Private**” and “**Public**” keys is and how it is related to the “**Https**” protocol. Then we will jump into the coding part, and discuss ways and methods of how programmers or developers handles information security during coding. After we will spare some time talking about how **mobile apps** handles it’s own apps security.

We all know what internet is. Internet for short is **W**orld **W**ide **W**eb. We use different kinds of web browsers to access the internet. The contents we access from or submit to must be base on two points. Web security is the way of how to secure these 2 points of connections, so as to avoid anoymous of accessing information between theses 2 points. During the early stage of WWW, we usually access the web site by using “**HTTP**” protocol. When we want to access for instance google home page, we must type in the url: “[http://www.google.com](http://www.google.com/)” in order to access. After years of time changing, if we want to access to google home page, we need to use the protocol “**HTTPS**”, this also becomes a standard for browsing web pages. What is the additional “**S**” character means, it means “**Secure**”. How this is done is base on the terms, “Private” and “Public” Keys. Below briefly discuss these 2 terms:

**What is private and public keys?**

In the simplest terms, private and public keys are like secret code pairs used to secure digital communication and data. They are a part of asymmetric encryption, where two different keys are used: one for encryption and one for decryption.

### Public Key:

* Shared with anyone: This key can be freely distributed.
* Used for encryption: People use your public key to encrypt information they send to you.
* Verification: It's also used to verify signatures made with the corresponding private key.

### Private Key:

* Kept secret: This key should never be shared.
* Used for decryption: You use your private key to decrypt information that others have encrypted with your public key.
* Signatures: You can use it to digitally sign documents, proving the information came from you.

#### A Real-World Analogy:

Imagine a public key as your email address (anyone can send you emails) and your private key as your email password (only you have access to read those emails).

Using both, you can ensure secure communication, authenticity, and integrity in digital interactions.

**By using public and private key terminology, please explain how web browsing process works.**

We will walk through a typical web browsing process involving public and private keys, focusing on the secure HTTPS connection.

### 1. **Requesting a Webpage:**

When you enter a URL into your browser (like https://example.com), your browser sends a request to the web server hosting that site.

### 2. **Server Responds with a Certificate:**

The server responds by sending its public key certificate. This certificate includes the server’s public key and is issued by a trusted Certificate Authority (CA), ensuring the authenticity of the server.

### 3. **Browser Verifies Certificate:**

Your browser verifies the certificate against a list of trusted CAs. If the certificate is valid and trusted, the browser proceeds. If not, you might see a security warning.

### 4. **Key Exchange:**

* Session Key Generation: Your browser generates a session key (a symmetric key for fast encryption).
* Encrypting with Public Key: The session key is encrypted using the server’s public key (from the certificate) and sent back to the server.

### 5. **Server Decrypts Session Key:**

The server uses its private key to decrypt the session key. Now, both the browser and the server share the same session key, which they will use for encryption.

### 6. **Secure Data Transmission:**

Using the shared session key, the browser and the server can now securely exchange information. This ensures that any data sent between them is encrypted and protected from eavesdropping.

### 7. **Browsing Continues:**

You can now securely browse the website, knowing your data (like login credentials, personal information, etc.) is protected by encryption.

Now that we have the basic understanding of how web browsing works, as said; programmers and developers also needs to program the code in order to adapt the changes.

1. Generating **Session Keys** on the client program, for example by using **JSON WEB TOKEN** or other forms of token generation.
2. Send the **Session keys** together when making a **Http Request** to the web server in the “Authorization” header field.
3. On the Server Side retrieve the **Session keys** from the Http Request method “Authorization” header field.
4. Now the Server and Client share the same “**Session Keys**” and can communicates in between.

How mobile apps works is a different approach. We don’t have a secure connection or linkage between the Apps and the Server, there is no such things like “**Https**” protocol. How this could be done is mainly base on how you program or handle data from your code. For instance, if you are working on a banking app, I assume you need to deal with transaction data between your app and the banking server. As mentioned before we can make use of “**Session Keys**” generate by the client app (banking app) and encrypted it to the “Http Request” method. Why “**Http Request**” method? as discussed this metod is usually works for the **web**. Yes! But that is the only way to secure your data. When your mobile app needs to deal with data or transaction from the banking server, In between the banking server (you can think of the server consists of database, images storage, videos storages...etc) and your client app, you need to setup a REST FUL Api Server. The RESTFUL Api Server acts a middleware Server in between the client app and the server. For web browing we use **Web server**, for data transaction we use **RestFul Api Server**. Just like web server, in order to connect to the Api Server, you also need to use the “HTTP/S” protocol. The same encryption methodologies can be used for securing the data like the Web.

Not only we can secure data transaction in between the app and the server. The app you build also is being digital signed and verify in order to be downloaded from the “Google Play Store” or “App Store”. The android app for instance needs to be digital signed when build and will generate an digital signed APK File, Google Play Store only accept digital signed APK file inorder for user to download from their platform.