Implementation of PSK-encrypted handshake protocol via Push Server for data transfer and communication between devices

Choiman1559 (cuj1559@gmail.com)

Chapter 1. Getting in

In this study, we study the push server, especially in applications using FCM (Firebase Cloud Message), first, to extend the use of FCM, which is mainly limited to server-to-device communication, to device-to-device communication, and second, to To improve security, a network architecture was built by designing and producing a handshake protocol by applying an algorithm similar to TLS-PSK (Transport Layer Security Pre-shared key) , Its effectiveness was verified by comparative analysis with other communication technologies such as Bluetooth and WiFi-Direct.

Chapter 2. Prior Research

2.1 TLS- PSK

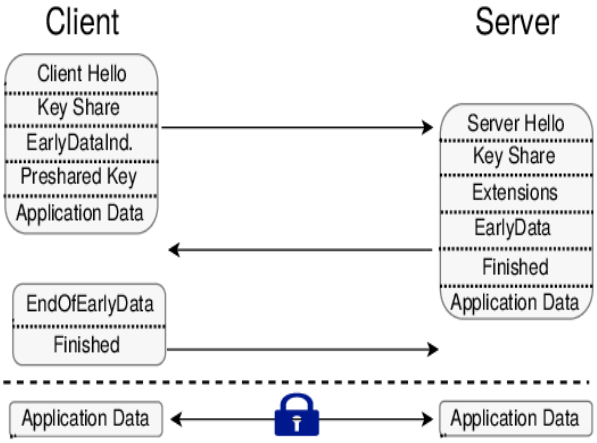


Figure 1. TLS architecture overview [1]

TLS is an abbreviation for Transport Layer Security, and is a name used when SSL (*Secure Sockets Layer)* is standardized. Also, PSK is an abbreviation of Pre-Shared Key, which refers to a symmetric key that is pre-shared between two parties for communication that requires security. At this time, TLS-PSK refers to establishing a TLS connection using PSK and is documented/standardized as RFC 4279.

The following describes the TLS handshake process shown in [Figure 1].

First, the client sends the TLS Version and one or more cipher-suites in the ClientHello message. Therefore, the server sends the TLS Version to be used and the cipher-suite to be used in the ServerHello message, and sends a PSK identity hint about which PSK the client will use in the ServerKeyExchange message along with ServerHelloDone. Finally, the client selects a PSK to use as a PSK identity hint included in ServerKeyExchange. Also, read the PSK Identity Hint received from the Server and check if it is valid. On the other hand, the server sets the PSK Identity Hint and callback to be used. This callback checks if the PSK identity is valid, and if there is a PSK identity, puts the pre-shared key in the SSL function argument, and returns the length of the PSK. This completes the handshake.

2.2 Firebase Cloud Messages

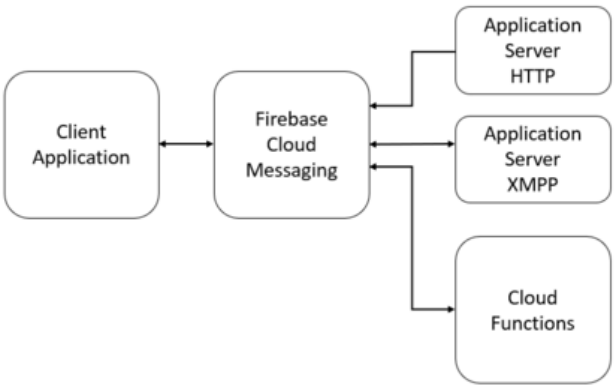


Figure 2. Overview of the FCM structure

The FCM architecture includes three components: a FCM connection server, a trusted environment with an application server based on HTTP or XMPP and cloud capabilities, and a client application. Sending and receiving messages requires a secure environment to build, forward, and send messages, as well as a server and an iOS, Android or web client application to receive the messages. FCM can also deliver targeted messages to applications through three methods: a single device, a group of devices, or a device subscribed to a topic. At this time, the app operator composes and sends a target message to a specific user group in the web-based 'notification composer'.

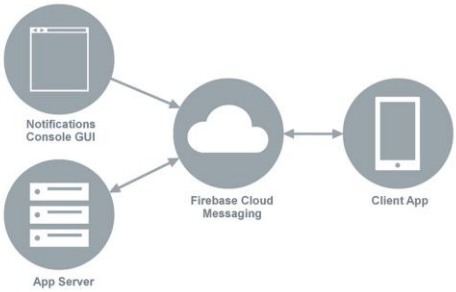


Figure 3. FCM process

To implement FCM, you must first register a device and configure it to receive messages from FCM. First, the client application instance is assigned a registration token or FCM token issued by the FCM connection server that will

provide a unique identifier to the instance and registers it. The app instance can then send and receive downstream messages. Downstream messaging means sending push notifications from the application server to the client application. This process includes 4 steps. First, a request for the message is sent to the FCM backend after the message is generated in the notification composer or other secure environment. Second, the FCM backend receives and accepts message requests, prepares messages for each specified topic, generates message metadata such as message IDs, and sends them to the platform-specific transport layer. Third, the message is sent to the online device through a platform-specific transport layer. The platform-level transport layer is responsible for routing messages to specific devices, handling message delivery, and applying specific configurations to the platform. Fourth, the client application receives notifications or messages through the device.

An inherent security issue with FCM is the potential exploitation of server keys stored in FCM's Android application package (APK) files. If exploited, it could distribute push notification messages to all users on the Firebase platform. GCM, the predecessor of FCM, previously reported security vulnerabilities that resulted in phishing and malicious advertising activity. [2]

Chapter 3. Design and Implementation

3.1 Network Flow Design

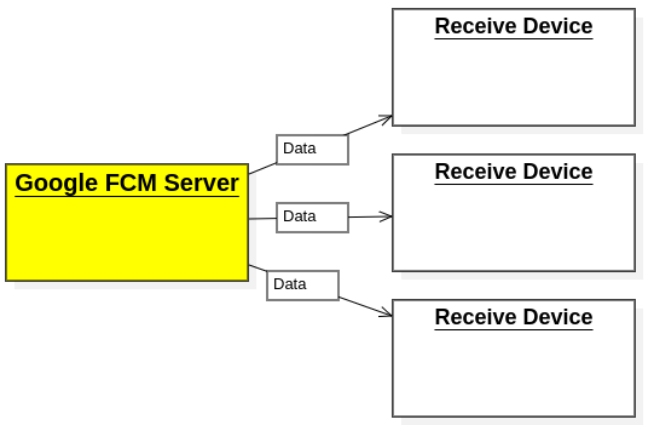


Figure 4. Traditional data flow

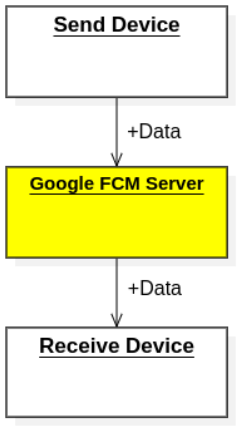


Figure 5. Target data flow

In general, FCM is designed and implemented on the premise of simplex communication, in which data is unilaterally transmitted from the server to the target device (Figure 4), so it is not suitable for duplex communication including communication between devices. Can not do it. To improve this, this study focuses on the fact that it is possible to use the Rest API for FCM to send a message to the target (registration token, subject or condition, etc.) After sending JSON-formatted data to the FCM server using an HTTP networking library such as Volley, the FCM server is used as a relay server so that the FCM server finally transmits the data to other clients to enable bidirectional communication (Figure 5). and the protocol to actually implement it was produced as follows.

3.2 Handshake Protocol Structure

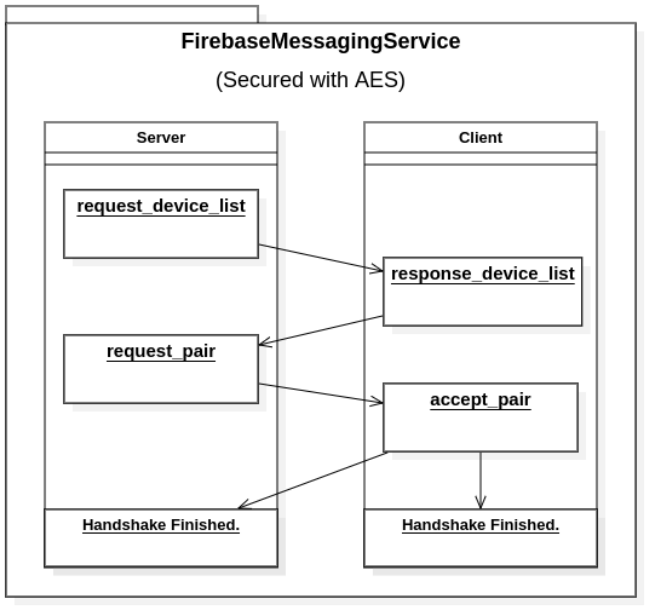
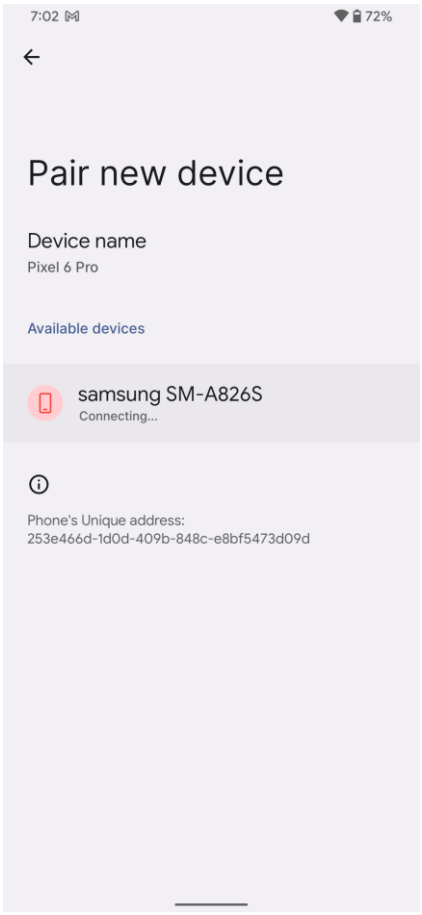


Figure 6. Handshake protocol architecture overview

In order to improve the security vulnerabilities of the aforementioned FCM and implement end-to-end communication, we designed the handshake protocol as follows. When the device to register other devices is called the server and the target device to be registered is called the client (Figure 5), first, *request_device_list* from the server is indiscriminately applied to a large number of unspecified devices that have subscribed to the same FCM topic [3]. It sends a call to request information about which device is listening to the signal and is ready to proceed with the handshake process.

Next, the client that has received the *request_device_list sends the client's device name and Unique ID to the server through response_device_list* , so that the server can recognize and distinguish the client, which is similar to TLS's *ClientHello .* Then , upon receiving *response_device_list* from the server, the included name and ID are added to the connection wait query (Query), and a list of connectable devices is displayed to the user (Figure 7), so the user can choose which device to handshake with. , which is similar to *ServerHello of*

TLS. When the user selects a device to proceed, it sends a *request_pair* signal to the client, and the server finally requests a connection to the client. Including *accept_pair* is sent, and when the server receives it, the included data is read. If the user decides to connect with the client, the client information is stored in the server. Otherwise, the received client information is deleted and the handshake is completed. On the other hand, in relation to security, the entire process described above is protected by the military-grade AES algorithm [4], and the details of encryption are described again in relation to technical matters in Chapter 3.3.1 of this paper.



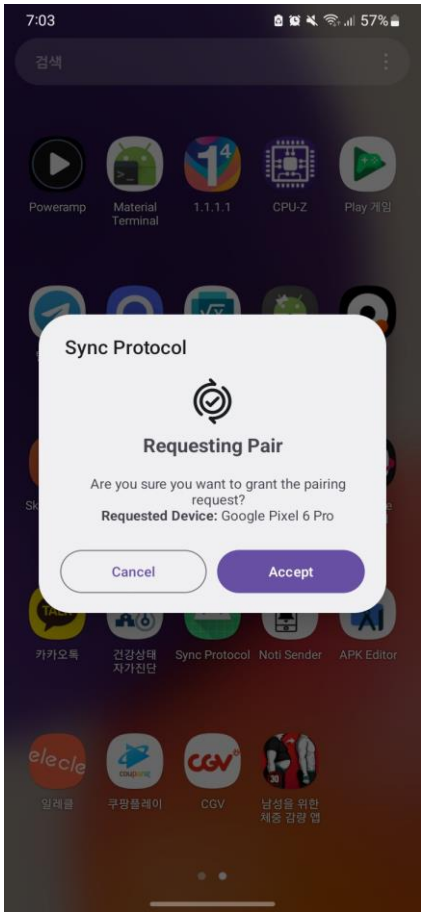


Figure 7. A scene from the server showing the user a list of connectable devices (left)

Figure 8. A scene where the client asks the user to connect to the server (right)

3.3 Technical Specifications

This paragraph describes the specifications necessary for the implementation of this protocol.

Before the description, this protocol adopts JSON (JavaScript Object Notation) as the data type, and sends data to the server through Rest-Api. After that, the client that receives the data processes the data through *FirebaseMessagingService* , and the details are as follows.

3.3.1 Data structure

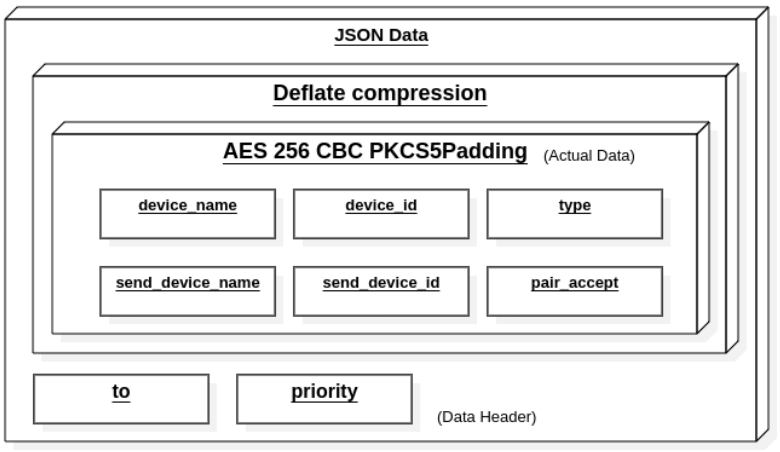


Figure 9. Schematic diagram of data sent and received by the server

First, the JSON object containing the data required for actual communication is encrypted with AES-256 CBC. Then , for data integrity authentication, HMAC hash generation is performed and the result is combined with the encryption result (encrypt-then-MAC). After that, in order to comply with FCM's maximum outgoing payload of 4kb, the data is reduced as much as possible using the Deflate compression algorithm, and the data along with header data is stored in JSON (Figure 9) and sent to the FCM server through Rest-Api. The form of the actual data described is as follows:

```
{
    "to":"<Device FCM topic>",
    "priority":"high",
}
```

Header:

to : Subject identifier handled internally by FCM.

priority : Android WakeLock related items handled internally by FCM.

```
{
    "data":{
    "encrypted":"true",
    "encryptedData":"<Encrypted&Compressed Raw data>"
    }
}
```

When encrypting data:

encrypted : Whether the data is encrypted or not. Represented as a boolean type.

encryptedData : A string in which data required for actual communication is encrypted.

At this time , *encryptedData is binary data that* is encoded as an 8-byte array and serialized through *Base64* and transmitted. When decrypted, it is divided into three parts and processed. (Figure 11) First, the 0~15 byte area is the IV key used for AES decryption, and is generally generated through pseudo-random numbers. Second, the 16-47 byte area is the SHA hash used for data

integrity authentication, and the stream created after AES encryption and the IV key are combined through the HMAC algorithm.

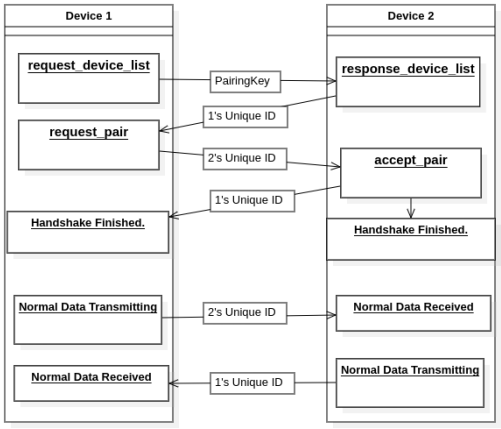


Figure 10. Schematic diagram of the key used for HMAC authentication when sending data

At this time, the key used for HMAC authentication uses the unique ID of the sending target device exchanged with each other during the pairing process. Authenticate. (Figure 10)

The last 48-4096 byte area is the result generated after encryption. The size of this data is variable depending on the size of the original data.

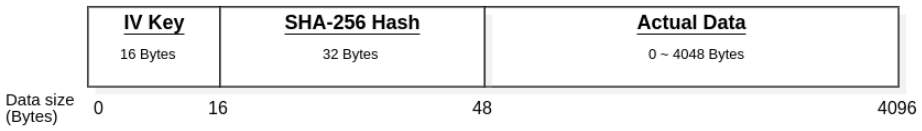


Figure 11. Final Encrypted Binary Data

When decrypting data:

```
{
    "data":{
    "device_name":"<actual Device name>",
    "device_id":"<actual Device id>",
    "type":"<data type>",
    "pair_accept":"<whether user accepted pair>",

    . . .

    }
}
```

device_name : The name of the device that sent the data. Usually expressed as "manufacturer + model name". (Example: Google Pixel 6 Pro)

device_id : A unique ID to identify the device that sent the data. Various methods such as GUID, Firebase IID, Android ID, WiFi Mac ID, etc. can be used for this item.

type : The type of data received. It may be various values according to need, such as "pair|request_device_list", "pair|response_device_list", "pair|request_pair", "pair|accept_pair", etc.

3.3.2 Processing after receiving data

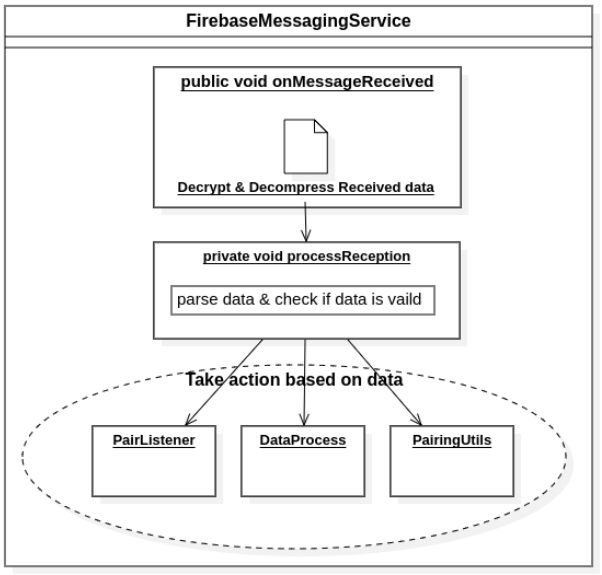


Figure 12. Process after receiving data

onMessageReceived **method** of the class that extends *FirebaseMessagingService* from the FCM server , the client decrypts the encrypted data, then sends the decrypted data to the *processReception* function to check if the data is valid, and then, if necessary, Various Listeners and Utility Classes
call to do the job. (Figure 12)

3.3.3 Modular structure

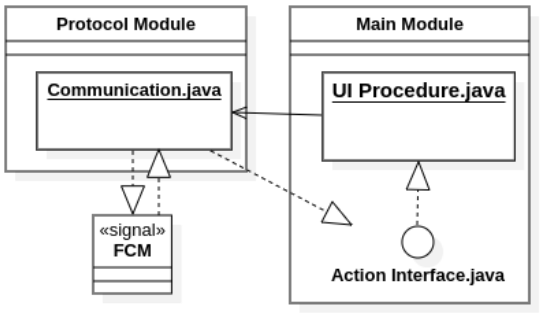


Figure 13. Modular structure overview

In addition, the data processing procedure and the UI processing procedure are separated by module unit, and the UI module calls the data processing protocol module at first to start and progress communication with FCM. Apps that need to apply the protocol by applying the modular design as a whole, such as proceeding through classes and listeners that inherit the predefined Interface from the UI module, and efficiently sending and receiving data through separate object wrapping By making it easily portable and available for use, it can provide time and economic advantages to the subject who uses it. In addition, by using this structure, it can be easily reflected when it is necessary to implement communication using a push server other than FCM in the future.

Chapter 4. Results analysis

4.1 Data transfer rate

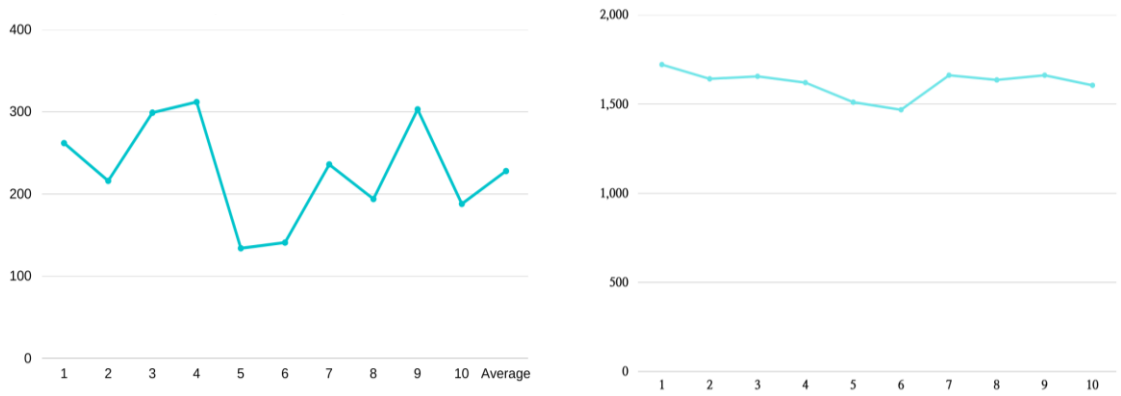


Figure 14, 15. Time required for unidirectional/bidirectional data transmission/reception (unit: ms) (Lower is better)

Performance test evaluation by measuring the time it takes to transmit/receive 4KByte sample data to each other 10 times with two devices at a distance of about 5m in an upload/download speed of about 94.4mbps, ping of 8ms, and IEEE 802.11ac connection environment As a result, it took an average of 228

ms in one direction and 1617 ms in both directions, resulting in a speed of about 20 ~ 140 kbps.

4.2 Data Reachable Distance

Protocol name	maximum reachable distance
Bluetooth 5.0 LE	up to 400 m [5]
WiFi Direct	up to 220 m [6]
Protocol presented in this study	no limit

Table 1. Comparison of maximum reach by communication technology

Unlike other communication technologies that are significantly limited by physical distance due to the unique structural characteristics of this protocol operating on FCM, it is theoretically shown that data communication is possible without any restrictions anywhere in the environment where the Internet is connected. can be considered a significant advantage. Also, to prove this in practice, after setting the experimental conditions to be the same as those presented in Chapter 4.1, changing the physical distance between the devices to about 115km based on the straight-line distance from Anyang to Taean, the result of the experiment showed that communication between devices was possible Conclusions have been drawn, which can be presented as evidence to support the above argument.

Chapter 5. Conclusion

If data communication is implemented in FCM without using such a protocol,

data that should be transmitted to a specific device may be unintentionally transmitted to multiple devices indiscriminately due to the characteristics of FCM, or data may be leaked due to greatly reduced security There are problems such as this. In that sense, this protocol is expected to be a solution that can solve more than a certain part of such problems. In addition, since this protocol was designed with not being dependent on a separate platform in mind, it is highly portable, simple to implement, and above all, excellent data exchange with advantages such as being independent of distance compared to similar communication protocols. It can be said to be a protocol, and it is expected that it will be usefully used in the future to create apps whose main purpose is data exchange between devices, such as Samsung 's Smart Switch, or to create messenger apps such as Telegram. However, compared to other communication protocols such as Bluetooth or WiFi Direct, the speed is slightly behind, so there may be some difficulties when transferring large files. It seems that it can be solved by going through a network) server, and adopting a symmetric key encryption method that disables all security once the key is stolen may be a flaw in security, but a more secure asymmetric key method through additional research and development in the future It seems that there is room to improve this disadvantage by replacing symmetric key encryption with RSA, etc.

Chapter 6. References

[1] M. Cebe and K. Akkaya, "A Replay Attack-Resistant 0-RTT Key Management Scheme for Low-Bandwidth Smart Grid Communications," *2019 IEEE Global Communications Conference (GLOBECOM)* , 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9013356.

[2] "Firebase Cloud Messaging Service Takeover: A small research that led to 30k$+ in bounties", https://abss.me/posts/fcm-takeover/#defining-impact

[3] "Topic Messaging in Android", https://firebase.google.com/docs/cloud-messaging/android/topic-messaging

[4] National Institute of Standards and Technology, " Security Requirements for Cryptographic Modules" (FIPS 140-2)

[5] Wikipedia, "Bluetooth" https://en.wikipedia.org/wiki/Bluetooth

[6] Pejman Roshan & Jonathan Leary, " 802.11 Wireless LAN Fundamentals" P.16

Chapter 7. Appendix

Full source code:

(Android & Java) https://github.com/choiman1559/RemoteSync/tree/module

(Electron & NodeJS) https://github.com/choiman1559/RemoteSync-Node