Development of Digital Certificate Management System on iOS Devices to Address Certificate Agility Costs in Certificate Pinning Mechanism

Daru Bagus Dananjaya
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
darubagus@gmail.com

***Abstract*—In this research, a digital certificate management system has been developed on iOS devices to address certificate agility costs in certificate pinning. Certificate pinning is a mechanism that matches digital certificates in software with digital certificates on a remote server, allowing communication to occur only between trusted parties to prevent man-in-the-middle attacks. However, the certificate pinning mechanism has a drawback known as certificate agility costs, which is a process that developers must regularly perform to update digital certificates in the application bundle to remain synchronized with the versions on the remote server. The management of digital certificates on local devices is accomplished by utilizing a different remote server to dynamically store fingerprints of the managed certificates. By employing this approach, the application can periodically update to stay up-to-date with the list of fingerprints on the remote server. In the final development phase, functionality testing of the system and usage testing on native software on the iOS platform were conducted. Based on the testing results, the system can address certificate agility costs in certificate pinning, although there are still administrative tasks that developers need to perform periodically to keep the list of fingerprints on the remote server up-to-date. This solution can eliminate the risk of adverse user experiences when users do not update during certificate rotation, thus preventing the application from becoming unusable. Additionally, it protects software from man-in-the-middle attacks conducted through SSL Proxying with the Charles Proxy tool.**

Keywords—Digital Certificate Management System, iOS, Certificate Pinning, Certificate Agility Costs, Man-in-the-middle Attack

I. INTRODUCTION

With the advancement of smartphone hardware, more and more people are becoming aware of the presence of crucial and sensitive data within these devices. Smartphone users are increasingly concerned about how their data is managed and transmitted. With the utilization of open networks to access various services and as a mechanism for communication between devices, threats aimed at exploiting smartphones have emerged. Some of these threats include denial of service, man-in-the-middle attacks, privacy breaches, among others [1].

Almost all software installed on devices requires network calls to obtain or store information on servers. When software attempts to establish a connection to a server, it is unaware of which digital certificates can be trusted and which cannot. Therefore, the software relies on digital certificates present in the iOS Trust Store or the prebuilt Android CA on the hardware. The problem arises when a third party can generate self-signed digital certificates and inject them into the Trust Store. This can create security vulnerabilities, enabling third parties to conduct man-in-the-middle attacks and intercept data transmitted to and from the software.

To mitigate existing threats, an additional security mechanism at the application layer is needed to ensure that all data transmitted between the server and the client remains private and intact. In this context, the technique of certificate pinning, a process to match a host with its digital certificate or public key, can be implemented to provide additional security at the application layer. This ensures that the software only trusts predetermined digital certificates or public keys. While certificate pinning is considered effective in thwarting MiTM attacks, it also has limitations.

Certificate pinning can pose challenges related to digital certificate management—a process that developers must undertake to update copies of digital certificates within the application bundle and potentially result in a poor user experience. Additionally, certificate pinning has security vulnerabilities as it can be bypassed through jailbreaking, a modification to a device made to remove file system access restrictions imposed by the device manufacturer.

In this research, we will discuss how the weaknesses of the Certificate Pinning mechanism on the iOS platform can be eliminated while still protecting users from MiTM attacks. A case study will be conducted on devices with the iOS operating system. iOS was chosen due to a poll conducted, where out of 14,000 respondents, approximately 23.11% stated that they do not enable the auto-update feature for applications and wish to have full control over application updates on their devices [2].

II. Related Study

A. Open Web Application Security Project (OWASP)

The Open Web Application Security Project (OWASP) is a community-based nonprofit organization aimed at creating standards related to security in web-based applications [3]. OWASP has a methodology called the Top 10, which identifies the most common vulnerabilities in web applications. The organization develops a Top 10 list of security threats for web, mobile, and IoT devices [4]. Based on the available list, this research will focus on the Top 10 mobile applications. Table 1 shows the top ten security threats in the mobile platform based on the latest update as of December 2016.

Table 1 OWASP Top 10 Mobile

Category	Issue
M1	*Improper Platform Usage*
M2	*Insecure Data Storage*
M3	*Insecure Communication*
M4	*Insecure Authentication*
M5	*Insufficient Cryptography*
M6	*Insecure Authorization*
M7	*Client Code Quality*
M8	*Code Tampering*
M9	*Reverse Engineering*
M10	*Extraneous Functionality*

Data in Table 1 indicates that improper platform usage occupies the top rank in OWASP security threats. This category encompasses security controls that are part of the operating system [5]. However, at the third rank, there is a threat related to insecure communication, making it a topic worth discussing. This category encompasses Certificate Pinning implementation.

B. Digital Certificate

A digital certificate is a combination of a statement and a digital signature of the related statement. In a network connection, users of a public key need to have confidence that the associated private key is owned by the correct subject. This confidence can be obtained through a public key certificate, which is a data structure that links a public key value to a remote subject [6]. This relationship is authenticated by using a certificate authority (CA) to sign the existing digital certificate. The CA can establish certificate ownership through a challenge-response protocol. A digital certificate has a limited lifespan, and this information is contained within the signed data of the digital certificate. Because the digital signature and timeliness of a digital certificate can be verified by user entities, digital certificates can be shared over an insecure communication network. The "X.509 public key digital certificate" is a commonly used type of digital certificate [7].

To perform a digital signature, the data to be signed must be encoded using the ASN.1 distinguished encoding rules (DER) format. ASN.1 DER is a tag, length, and value encoding system for each element in the digital certificate [6].

C. Transport Layer Security

Secure Socket Layer (SSL) is a cryptographic security protocol developed by Netscape around 1990. It is widely used to provide confidentiality, authentication, and message integrity in communication [8]. SSL offers three main security services: confidentiality through data encryption, message integrity through a message authentication code (MAC), and authentication through digital signatures (Hickman, 1995). The SSL protocol was succeeded by Transport Layer Security (TLS) as the standard protocol for securing internet connections.

The use of TLS enables authentication between two parties: an authenticated server and an anonymously unauthenticated client. Authentication can occur between the client and server through digital signatures. Currently, most implementations of digital signatures use certificates (e.g., X.509 standard) or shared keys. In the case of certificates, a Certificate Authority (CA) is needed to ensure proper signing. In contrast, shared key authentication can be performed using cryptographic algorithms like Diffie-Hellman, starting from SSLv3.00, TLSv1.00, and newer versions [9].

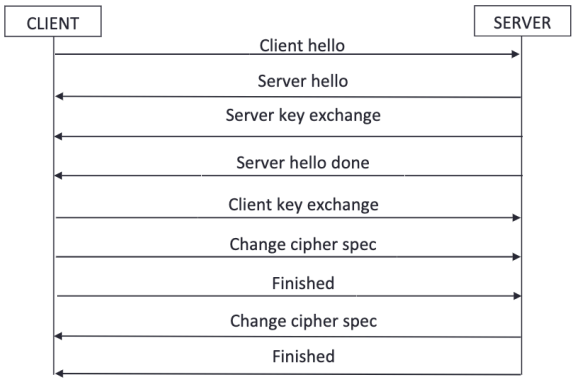

Figure 1 TLS Handsake [10]

The SSL/TLS protocol uses a handshake process consisting of: (1) The client sends a ClientHello message to the server, containing a list of compatible cipher suites and a client_random value; (2) The server responds with a ServerHello message, containing the server's chosen cipher suite and a server_random value. Additionally, the server sends a session ID, serving as a reference for future sessions; (3) The server proves its identity by sending its digital certificate, and the client verifies this identity using the public key of the CA that issued the certificate; (4) Once verified, the server sends a ServerHelloDone message; (5) The client sends a second random number called pre_master_secret, encrypted with the server's public key. The client_random, server_random, and pre_master_secret are input into a function to generate the master_key. The master_key, client_random, and server_random are used to derive the secret key and MAC key. On the server side, data received from the client is decrypted, and the server generates the same secret key and MAC key as on the client side; (6) The client sends a

ChangeCipherSpecification message, indicating that encryption of messages after this point will be performed using the generated secret key; (7) The client sends a HandshakeFinished message, signaling readiness to enter the data transfer phase; (8 & 9) The server sends ChangeCipherSpecification and HandshakeFinished messages for the same purpose as (6) and (7). This process is illustrated in Figure 1.

D. Certificate Pinning and Public Key Pinning

Pinning technique, commonly known as HTTP Public Key Pinning (HPKP), has emerged as an effort to strengthen software security against MiTM (Man-in-the-Middle) attacks[11]. SSL pinning has been widely used to enhance the security of SSL/TLS communication in non-web mobile software[12]. Pinning ensures that the software only communicates with predefined servers [13].

SSL pinning, typically divided into two phases, in the first phase, the client initiates communication with the server through a client hello. After receiving the client's message, the server responds with its status through a server hello. Then, the client requests the server's digital certificate, and the server responds with information from the digital certificate and its public key. In the second phase, the client verifies the previously received digital certificate from the server. When the client receives a message from the server, it checks its authenticity using the server's public key stored on the client. If the public key matches, the client negotiates or sends a packet signed with the public key. If the verification process fails, communication is halted, and no packets are sent to the server.

In general, an application will include a list of received digital certificates or the content of those certificates, including subject distinguished names, fingerprints, serial numbers, and their public keys. If the application explicitly defines a list of accepted CA (Certificate Authority) digital certificates, periodic updates are required when CA certificates change or expire. Despite offering protection against MiTM attacks, SSL pinning practices have been debated due to the overhead they introduce, known as certificate agility costs. Additionally, HPKP implementation has been deprecated.

Pinning has proven to be a good preventive measure against MiTM attacks [14][15]. The use of SSL pinning is becoming common to enhance the security of native SSL/TLS implementations, thus guarding against MiTM attacks. However, SSL pinning still has security vulnerabilities, which can be bypassed, although most bypasses are possible only on jailbroken iOS devices or rooted Android devices.

SSL pinning mechanisms can be considered relatively straightforward: they associate a host that the application will connect to with a certificate or public key that meets IETF X.509 cryptography standards. When the association between the host and certificate is successful, a secure connection is established, and the digital certificate is pinned to the host. A secure connection is established when API calls are made to instances with certificates listed in the pinned certificate list.

Typically, received or pinned digital certificates are embedded in the application bundle during software implementation. Adding pinning mechanisms adds an extra layer of security by making it difficult for third parties to disable the pinning system. However, there is an alternative, which is pinning during the initial connection establishment—a process known as key continuity. But using key continuity in this manner can potentially increase the risk of attacks during the host-client connection establishment [16].

There are two SSL pinning options available: (1) certificate; or (2) public key [16]. If pinning is done with the public key, there are two additional options: (a) subjectPublicKeyInfo; or (b) pin one of the more concrete parameters like RSAPublicKey or DSAPublicKey. In option (1), the digital server certificate is embedded in the application bundle, and during runtime, the software compares the remote certificate with the embedded digital certificate. For option (2), the public key from the certificate is hardcoded into the codebase as a string, and during runtime, the software compares the public key in the remote certificate with the pre-hardcoded public key value.

E. TLS vs. Domain Name System Security Extensions (DNSSEC)

There is a debate about what is better, using Transport Layer Security (TLS) or DNS Security Extensions (DNSSEC). Ideally, both should be used because they address different issues, using different methods, and operate on different data [17]. SSL/TLS encrypts data and authenticates websites, while DNSSEC validates digital signatures obtained from DNS servers.

In this context, SSL/TLS is a system used to encrypt data and authenticate the sender – in this case, the accessed website. To facilitate this encryption, the entity running the website must obtain an SSL Certificate. This digital certificate is used to validate the identity of the communicating parties, thus establishing a secure channel. Through the established session, all data passing between the website and the browser is encrypted – to anyone intercepting network traffic and looking at message packets, it appears scrambled.

DNS Security Extensions (DNSSEC) does not interact with websites at all – everything happens behind the scenes before any web activity occurs [17]. When a computer uses DNSSEC to look up a website's address, it not only performs a regular DNS lookup but also validates the signatures returned from DNS servers. This activity occurs at all levels, from the "root server," through high-level domains (such as .org or .info), to the specific requested address (such as www.example.org). If all signatures can be validated, a valid response is sent to the browser to connect to the website, which may use SSL to do so. Both technologies play a crucial role in protecting online information and data integrity, but they focus on different aspects. Table 2 shows a comparison of SSL and DNSSEC from various perspectives and how they can be used together to enhance overall online security.

Table 2 SSL/TLS and DNSSEC Comparison

Criteria	SSL/TLS	DNSSEC
Purpose	Encrypts data and authenticates websites.	Ensures the integrity of DNS

Criteria	SSL/TLS	DNSSEC
		data by validating signatures.
Authentication	Uses SSL certificates issued by Certificate Authorities (CAs).	Authenticates DNS data by verifying digital signatures.
Data Encryption	Encrypts data transmitted between the browser and the website.	Does not encrypt data; only validates DNS data authenticity.
Protection	Protects against eavesdropping and data interception.	Protects against DNS data manipulation and spoofing.
Trustworthiness of CAs	Vulnerable to CAs issuing certificates without verification.	Relies on the chain of trust from root DNS servers to validate.
Self-signed Certificate	Some websites can use self-signed certificates, potentially making phishing easier.	Not applicable, as it doesn't deal with website certificates.
Limitations	Doesn't provide comprehensive protection against network snooping.	Limited to DNS data validation; doesn't secure web interactions.
Overall Security Benefit	Enhanced security when used together with DNSSEC for complete protection.	Provides security for DNS data but needs SSL for broader web protection.

F. *Man-in-the-middle Attack*

A Man-in-the-Middle (MiTM) attack is a type of computer security attack where an attacker inserts themselves between two parties communicating directly. In this attack, the attacker gains control over the supposedly secure communication flow between the two entities, enabling them to eavesdrop, modify, or inject data being transmitted between them[18]. Typically, the primary objective of a Man-in-the-Middle attack is to steal personal information from one party, such as login tokens and credit card numbers [18]. Figure 2 illustrates the scheme of a Man-in-the-Middle attack, showing that the intruder can intercept, send, and receive data intended for others without the knowledge of both parties [19].

Man-in-the-Middle attacks can be categorized into four types[19]. First, there are spoofing-based attacks, where a third party intercepts network traffic using spoofing tools to control transmitted data without the host's knowledge. Further, spoofing-based attacks can be divided into two subtypes: DNS spoofing, where a third party spoofs devices between endpoints, and ARP spoofing, where a third party directly spoofs endpoints or devices used by the victim. Second, there are attacks on SSL/TLS, where a third party positions themselves on the communication network between endpoints or victims. In SSL/TLS attacks, the third party establishes two separate SSL connections with the victim and relays data, as shown in Figure II.6. Third, there are attacks on the Border Gateway Protocol (BGP), where a third party redirects data packets to the desired destination. Finally, there are false base station attacks, where a third party creates a fake transceiver node and uses it to manipulate victim traffic.

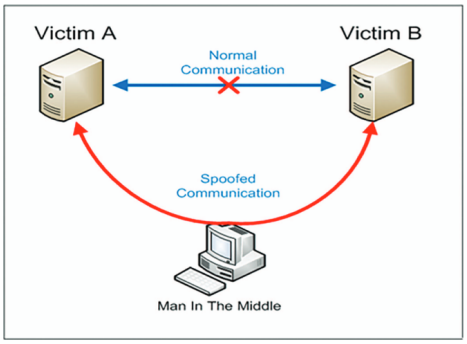

Figure 2 Man-in-the-middle Attack Scheme[10]

G. *Security in iOS*

Every iOS user must agree to the End-User License Agreement (EULA) when first running an application. The Apple operating system does not provide external security analysts with access to analyze the security of the source code but rather employs a "security through obscurity" model that prevents others from reverse engineering. However, this approach is considered unsuitable for large-scale applications [20].

In software development within the iOS ecosystem, developers are encouraged to use available frameworks at higher levels because higher-level abstractions are object-oriented. Nonetheless, developers still have access to features such as sockets and threads [20]. From a security perspective, Apple asserts that with a highly integrated configuration between software and hardware, activity validation can be performed at all layers within the device's architecture [20].

H. *URLSession in iOS*

When accessing a URL via an HTTPS connection using URLSession, what happens behind the scenes is that the connection occurs over HTTP on top of TLS (Transport Layer Security). Therefore, when accessing the same server for the second time and subsequent times, the client will not receive a challenge like the first connection because it will use the session that has already been established.

The initiation process of a TLS session is computationally expensive as it involves a number of calculations with large numbers. Consequently, TLS has mechanisms to avoid this work for every connection. A TLS connection can either establish a new session or attempt to use an existing session. Therefore, TLS clients generally improve their performance by utilizing a cache of existing sessions.

iOS and macOS devices utilize TLS cache for all connections made to remote servers [21]. TLS cache keeps

connections alive for a certain period to enhance the response time of HTTPS requests. However, developers do not have direct access to the process of caching these connections, for example, developers cannot override or close connections that have already been established [21].

III. Proposed Solution

The data exchange process in native mobile applications with a backend server is typically carried out through an HTTP secure transfer protocol. With the use of open networks to access various information, there are parties with the intention to exploit devices and data. An API call over the HTTP secure transfer protocol can be susceptible to Man-in-the-Middle (MiTM) attacks performed with tools like Charles Proxy. Therefore, a security mechanism is needed to protect the communication between the client and the backend server, and one way to implement this is through certificate pinning.

Based on the background explanation, there is currently a mechanism called SSL pinning, more commonly known as certificate pinning. Certificate pinning ensures that the Mobile SDK checks the digital certificate of the server it is connecting to against a list of digital certificates recognized by the device. This mechanism prevents the application from third parties attempting to use fake digital certificates to compromise trust between the user, developer, and the application.

While certificate pinning can enhance application security, it has sparked debate among developers due to the additional effort required for certificate management. To address this issue, a certificate management mechanism is needed to implement certificate pinning with minimal maintenance. This approach involves utilizing a different remote server to store fingerprints of dynamically managed certificates. By doing so, the application can periodically update digital certificate data to stay current with the fingerprint list from certificates on the remote server.

A. General Solution

Based on the analysis of the solutions, a comprehensive solution has been designed, which includes a digital certificate management system and certificate pinning. This decision to implement certificate pinning instead of DNSSEC in the application is based on an evaluation of security objectives and specific challenges faced by the mobile application. Certificate pinning was chosen due to its ability to address specific security issues, as most applications prioritize data integrity and confidentiality during communication between the mobile application and the backend server.

By predefining trusted server certificates within the application, certificate pinning effectively prevents Man-in-the-Middle (MiTM) attacks by ensuring that the server's certificate matches its identity. This validation process makes it extremely difficult for attackers to intercept or manipulate data during transmission, thereby enhancing the overall security of the application.

Furthermore, certificate pinning provides immediate security benefits as it does not rely on DNS resolution, allowing the application to establish secure connections quickly. This rapid response is highly advantageous for applications requiring fast and secure data transfers.

The approach used for implementing certificate pinning will leverage a different remote server from the backend server to store a list of fingerprints. This remote server will store the list of fingerprints, which will then be downloaded. The following are the functional requirements of the system created:

1. Manage the list of fingerprints downloaded from the remote server, including scheduling updates and filtering invalid fingerprints.
2. Digitally sign all entries in the fingerprint list with a private key, then validate them using the public key during the data fetch process before storing them in persistent storage.
3. Provide certificate pinning mechanisms through fingerprint validation to enhance application security during the TLS handshake.

Based on the analysis, a solution has been designed, comprising a digital certificate management system and certificate pinning. Certificate pinning leverages a separate remote server to store fingerprints, ensuring secure TLS handshake validation. This approach enhances application security by managing and validating digital certificates effectively, addressing existing issues.

B. Functional Requirements

Based on the analysis of the solutions, a solution comprising a digital certificate management system and certificate pinning has been designed to address the existing issues. The design consists of two parts: the design of the digital certificate management library and the design of certificate pinning implementation.

Based on the design, additional functional requirements for the system have been identified. These functional requirements are the result of the development of the general solution after designing the modules within the digital certificate management system

1. The library can accept input such as serviceURL and public keys from the remote server to be connected.
2. The library is capable of downloading certificate data from the remote server.
3. The library can perform data storage and loading utilities to persistent storage.
4. The library can identify cases when the persistent storage is empty.
5. The library can validate certificates during communication with the real server.
6. The library can validate digital signatures.
7. The library can schedule certificate updates in persistent storage.
8. The library can perform data hashing.

The design process has resulted in additional functional requirements for the system, including configuration, data handling, validation, and scheduling capabilities within the digital certificate management system. These enhancements aim to improve the overall functionality and security of the system.

C. Remote Server

A separate remote server is required from the main backend server domain that will be accessed by the application. This is due to the TLS cache present in URLSession as explained before. In this research, the storage of digital certificate lists is demonstrated through a NoSQL JSON database hosted on my-json-server.typicode.com. In the implementation of this database, it is assumed that the ECDSA key pair belonging to the server has been generated. This key pair is used to sign the existing digital certificates for the present and the future. The signing process is intended to ensure the integrity and authenticity of the digital certificates, guaranteeing that they come from a legitimate source and have not undergone unauthorized data changes.

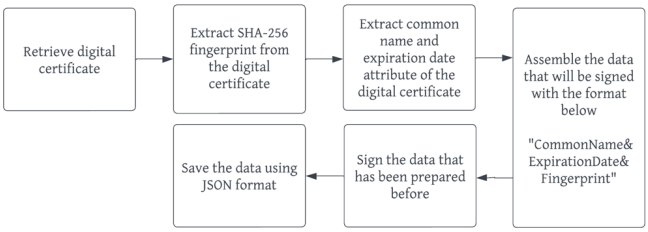

Figure 3 Data Extraction Flow from Digital Certificate

The preparation of fingerprint data begins with fetching digital certificates from the server to be targeted. In this research, digital certificates are retrieved from "openweathermap.org." These digital certificates are obtained in a format optimized for SHA fingerprint creation, specifically in DER format. The data will be signed using ECDSA; hence, a key pair using prime256v1 curve is prepared.

The next step involves calculating the SHA-256 fingerprint of the previously obtained digital certificate, after which the fingerprint data is encoded into a Base64 string. Attributes of the digital certificate, namely the common name (CN) and expiration date, are extracted. The CN attribute is obtained from the subject field, while the expiration date is acquired from the validity field and then converted into UNIX timestamp format.

Once all the components are obtained, a signature base string is constructed by combining the CN attribute, UNIX timestamp, and fingerprint separated by an "&" sign. Subsequently, a digital signature is created using the ECDSA private key. The obtained digital signature is then encoded as a Base64 string.

The final result of the preparation process is a JSON file that stores fingerprints, expiration dates, common names, and digital signatures of digital certificates. The JSON format is designed to accommodate multiple fingerprints to facilitate the storage and management of digital certificate data. The steps taken to prepare fingerprint data can be viewed in the flowchart diagram shown in Figure 3.

D. System Architecture

Based on the analysis conducted, the digital certificate management module is a key module in the library that serves as the orchestrator for digital certificate management processes and provides all services related to SSL certificate pinning. The overall system design can be viewed in Figure 4. Within this module, several other modules need to be added, namely: the crypto provider module, the secure data store module, and the remote data provider module.

Each of these modules has its own functionality. Here are the responsibilities of each module:

1. The secure data store module serves as a data storage service provider for certificate data in persistent storage. All sensitive data related to digital certificates used in the library solution will be stored in the Keychain, an encrypted database provided by Apple.
2. The crypto provider module functions as a cryptography service provider for the proposed library. This library uses the P256 curve for digital signatures and hash functions. The selection of this curve is based on its popularity for creating digital certificate fingerprints.
3. The remote data provider module serves as an HTTPS request service provider for retrieving fingerprint data from the remote server's database.

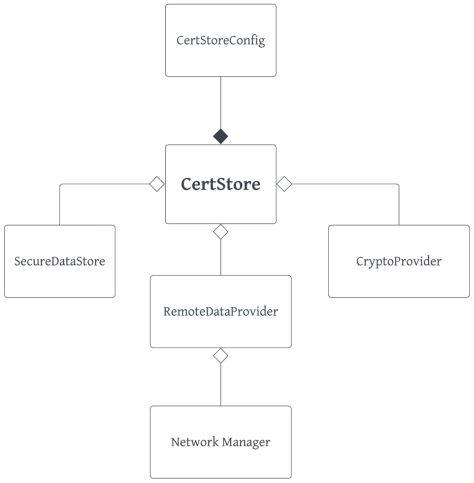

Figure 4 Digital Certificate Management System Architecture

The Certificate Store module includes supporting classes, such as the Certificate Store Configuration class. This class is created to enhance readability and facilitate library users in configuring various settings, including scheduling and fallback data.

E. TLS Certificate Validation

The SSL certificate validation or certificate pinning implemented in this library follows the standard practices of certificate pinning but with adjustments to work effectively within this library. The validation process will be implemented as an extension of the certificate store module. This library

offers several input options for certificate validation to accommodate different scenarios for library users, including:

1. Validation with input data of common name and fingerprint.
2. Validation with input data of common name and digital certificate data in .der format.
3. Validation with input URLAuthenticationChallenge.

Calling the validation method will result in return values as enumerations, namely trusted, untrusted, and emptyStore. This library employs a whitelisting approach, where by default, all connections are considered untrusted. For these three return values, the connection will only proceed if the return value is trusted. Otherwise, the TLS handshake in progress will be canceled. The workflow of the SSL certificate validation process can be seen in Figure 5.

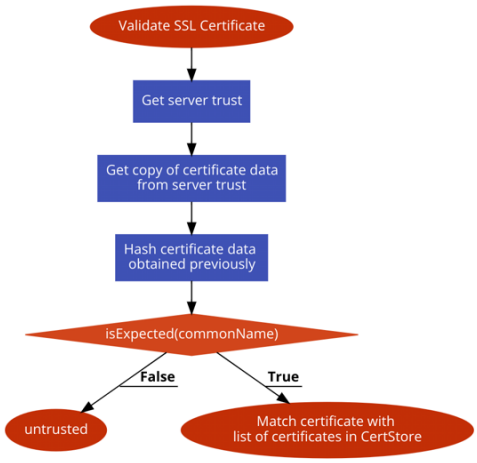

Figure 5 TLS Certificate Validation Flow

F. Fingerprint Update

In this library, the ideal fingerprint data update process should occur when the application is launched, before initiating HTTPS requests to the server and certificate pinning validation. The certificate update in this library provides two modes:

1. Direct Mode: This mode is chosen when all certificates on the device have expired or when the application is first opened (freshly downloaded from the App Store with no certificate fingerprint list in persistent storage). In this mode, the application halts all processes to wait for the fingerprint list download.
2. Silent Mode: In this mode, updates are not executed immediately but placed in the completion queue. The certificate store module then handles the update process in the background. The goal of this update mode is to avoid blocking the application when opened while keeping the fingerprint list up-to-date. The update frequency is determined automatically by the certificate store.

The retrieval of fingerprint data from the server is performed by the Remote Data Provider module. The mechanism for fetching fingerprint data from the remote server can be seen in Figure 6. When the request to the remote server is successful, the method that handles the processing of response data from the server to storage takes place end-to-end. If the request fails, an error message is issued, terminating the data update process.

The method for processing data from the remote server performs decoding of the JSON response obtained. Subsequently, the cache update is carried out with the newly received data. This process is performed by iterating through all the data in the array formed from the response. The certificate data update consists of several steps, which can be summarized as follows:

1. Iteration through all entries of the JSON response.
2. Conversion of entry values according to the supported format.
3. If the entry already exists in the device's persistent storage, it will be skipped. Entries are considered the same if the common name, expiration date, and SHA-256 value of the fingerprint are identical, preventing duplicate data and potential insertion of false data into persistent storage.
4. Validation of ECDSA for each entry to ensure that only valid and trustworthy fingerprint data is stored.
5. If step 4 succeeds, the entry is added to persistent storage. If step 4 fails, data processing is halted, and the remote server data update process is cancelled.

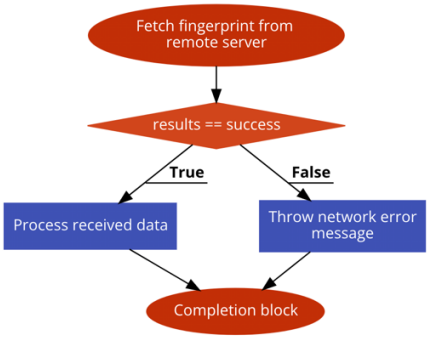

Figure 6 Fetching Remote Server Data Flow

If processing all acquired data is successful, the end of the fingerprint update process includes an additional check on the overall process status. If the results align, the fingerprint certificate data is sorted based on name and expiration date. After processing all fingerprint data, the next update is scheduled, and the process concludes with the storage of processed fingerprint data into persistent storage.

IV. RESULTS

At the end of the research, functional testing was conducted to evaluate the functionality of the developed library based on the defined functional requirements in Table III.3 using Unit Tests. Testing was carried out not only at the system level but also at the module level to ensure that the Certificate Store, Secure Data Provider, Crypto Provider, and supporting modules

could meet the system's functional requirements and identify potential issues that might arise at the component level. Additionally, testing was performed to ensure that the library could resolve issues that occurred.

Based on the test results, all standard test cases were successfully met. All test scenarios produced the expected results. Therefore, all the functional requirements of the system aimed at managing digital certificates in iOS software have been fulfilled. Here are the steps taken to address the issues:

1. The issue related to digital certificate agility in certificate pinning was resolved by adding a digital certificate management mechanism to retrieve and store digital certificate data from a remote server, which is then stored in persistent storage.
2. The issue related to the authenticity of digital certificates when stored on a remote server and data transport was ensured through digital signatures with the server's private key. The validation of these signatures is performed during the process of fetching digital certificate data from the remote server using the server's public key, ensuring that only authentic digital certificate data is processed.

Furthermore, smoke testing was conducted to test the functionality of the native software developed to test the digital certificate management system and certificate pinning from the previously created library. Testing on the software was performed to ensure that the library can perform its functions as required when used in software. Smoke testing was performed on a simple native iOS software with API call functionality to "openweathermap.org".

Based on the testing conducted, utilizing the previously created library, the software was able to prevent MiTM attacks through SSL Proxying, which in this case was carried out using the Charles Proxy tool. When sniffing was attempted on a connection using certificate pinning, the connection was terminated because pinning did not occur on the digital certificate as shown in Figure 7. However, when normal connections were established, communication with the server was deemed legitimate and continued with data transfer, allowing the software to receive responses from the backend server. Therefore, all the system's functional requirements, as described, were met when the library was used in software.

Based on the test results, the issue of digital certificate management arising from certificate pinning has been effectively addressed through the created system. The digital certificate management mechanism to eliminate certificate pinning overhead has been successfully implemented and achieved the goals of the research.

V. DISCUSSION

The evaluation was conducted by comparing the results of the proposed solution with the standard certificate pinning mechanism using URLSessionDelegate. The evaluation results will serve as input and recommendations for further development of the digital certificate management solution to address certificate agility costs in certificate pinning.

Based on the evaluation results comparing the standard certificate pinning conditions with the proposed solution in Table 3, it can be concluded that the development of the digital certificate management system on iOS devices can address certificate agility costs in certificate pinning. Although there are still administrative tasks that developers need to perform periodically to keep the fingerprint list on the remote server up-to-date, this solution can eliminate the risk of a bad user experience when users do not update during digital certificate rotation, thus preventing the application from becoming inoperative.

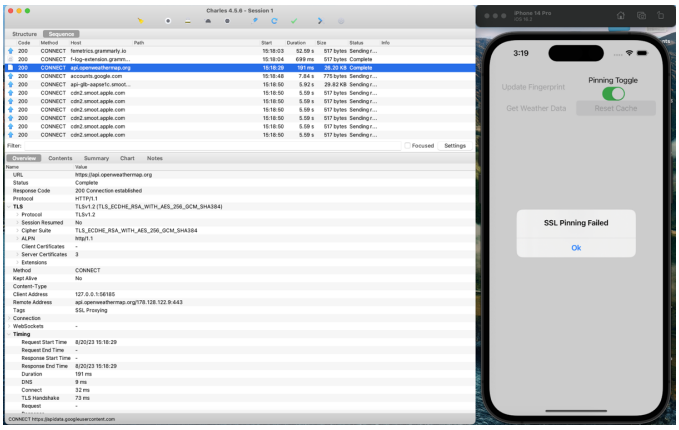

Figure 7 API Call with SSL Proxying

From a security perspective, based on testing related to simulating man-in-the-middle attacks using the Charles Proxy tool on native iOS software with API call functionality to a specific remote server, the results show that the implementation of certificate pinning in the library can prevent man-in-the-middle attacks. These results demonstrate that the level of security provided by the library is equivalent to the level of security provided by the standard certificate pinning mechanism using URLSessionDelegate. Therefore, it can be concluded that the man-in-the-middle attack prevention mechanism through certificate pinning provided by this library has been successful and fulfills the objectives of this research.

There are several limitations in the implemented results. First, as previously explained, when developers use URLSession to make network calls, the fingerprint list should not be stored on the same domain because developers do not have control over the TLS cache. In other words, developers cannot forcibly close established connections. Therefore, if the fingerprint list is placed on the same domain, the application will reuse connections that have been established during the process of downloading the list of digital certificates, rendering the certificate pinning mechanism ineffective.

Second, connections to the remote server used for downloading the fingerprint list should not be pinned. The fingerprint list must always be available and accessible, and securing it using pinning for the domain storing the fingerprint list will lead to a deadlock from the certificate store's perspective. Therefore, the process of downloading the fingerprint list should always be considered trusted.

Third, the library created assumes that the remote server storing the fingerprint data can always be trusted, so no additional security mechanisms, such as authentication-challenge, have been implemented. For further development, such mechanisms can be implemented during the process of retrieving fingerprint data.

Finally, the implementation of the library in this research is limited to native applications on the iOS operating system. However, the workings of this system can be applied universally. Therefore, for future development, the system can be implemented on the Android operating system by utilizing the previously explained system's operation.

Table 3 Comparison between Standard Certificate Pinning Mechanism and Proposed Mechanism

Aspect	**Standard Mechanism in Certificate Pinning**	**Certificate Pinning with Digital Certificate Management System**
Certificate Validation	Performed by comparing the digital certificate values in the application bundle to the digital certificate obtained from the server.	Performed by comparing the certificate obtained from the server against the list of certificates stored in persistent storage. The checks are related to the expiration date of the digital certificate, the common name, and the fingerprint.
Certificate Rotation	An update is required in the App Store with the new bundle of digital certificates.	Updates are required on the remote server's database that stores the list of digital certificates, but there is no need to update the application in the App Store.
User Experience	If users do not update the application when digital certificate rotation occurs, the application will become unusable due to failed certificate validation, resulting in an inability to communicate with the server.	User updates are not an issue because when a digital certificate change occurs, the application will automatically update the digital certificate data stored in persistent storage.
Developer Experience	With each digital certificate rotation, the developer needs to embed the digital certificate into the application bundle and then upload it to the App Store.	With each digital certificate rotation, the developer needs to add the replacement digital certificate data and its associated fingerprint to the remote server's database.

VI. CONCLUSION

A digital certificate management library was successfully implemented to address digital certificate agility costs in certificate pinning and prevent man-in-the-middle attacks. Based on the conducted tests, the mechanism for digital certificate management to eliminate overhead related to certificate agility in certificate pinning has been successfully implemented. All formulated test cases in the functional testing were met, ensuring that all functional requirements of the digital certificate management library have been fulfilled.

The certificate pinning mechanism involves comparing certificates obtained from the server with a list of certificates stored in persistent storage, conducting verification regarding the expiration date of digital certificates, common names, and fingerprints. Thus, the certificate pinning mechanism in the proposed system in this research can effectively prevent man-in-the-middle attacks, as demonstrated through simulations using Charles Proxy.

To further enhance security, an additional layer of security in the form of a random challenge can be implemented during the process of fetching fingerprint data from the remote server. This would make it more difficult for third parties to predict or replicate responses. Even if third parties intercept the challenge-response process, they cannot use the response for a different challenge since each challenge data is unique.

The digital certificate management system and certificate pinning for iOS devices have been successfully implemented to eliminate certificate agility costs and prevent man-in-the-middle attacks. For future developments, this mechanism can be adapted for Android devices using the system's existing framework created in this research.

REFERENCES

[1] R. Setyawan, A. A. Rahayu, K. F. Nur Annisa, dan A. Amiruddin, "A brief review of attacks and mitigations on smartphone infrastructure," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 852, no. 1, 2020.

[2] R. Ritchie, "Will you be using auto app updates on iOS 7? [Poll]," 2018. [Daring]. Tersedia pada: https://www.imore.com/will-you-be-using-auto-updates-ios-7-poll. [Diakses: 01-Sep-2023].

[3] S. Kim, H. Han, D. Shin, I. Jeun, dan H. Jeong, "A study of international trend analysis on web service vulnerabilities in OWASP and WASC," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5576 LNCS, hal. 788–796, 2009.

[4] F. J. Ramírez-López, A. J. Vaca-Varela, J. Ropero, dan A. Carrasco, "Guidelines Towards Secure SSL Pinning in Mobile Applications," hal. 238–244, 2019.

[5] OWASP, "M1: Improper Platform Usage," 2016. [Daring]. Tersedia pada: https://owasp.org/www-project-mobile-top-10/2016-risks/m1-improper-platform-usage. [Diakses: 10-Feb-2023].

[6] R. Housley *et al.*, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," *Netw. Work. Gr.*, 2002.

[7] L. Harn dan J. Ren, "Generalized digital certificate for user authentication and key establishment for secure communications," *IEEE Trans. Wirel. Commun.*, vol. 10, no. 7, hal. 2372–2379, 2011.

[8] K. E. . Hickman, "The SSL Protocol," *Netscape Communication*

Corp, 1995. [Daring]. Tersedia pada: https://datatracker.ietf.org/doc/html/draft-hickman-netscape-ssl-00.

[9] X. Gu dan X. Gu, "On the detection of fake certificates via attribute correlation," *Entropy*, vol. 17, no. 6, hal. 3806–3837, 2015.

[10] M. Alwazzeh, S. Karaman, dan M. N. Shamma, "Man in The Middle Attacks Against SSL/TLS: Mitigation and Defeat," *J. Cyber Secur. Mobil.*, vol. 9, hal. 449–468, 2020.

[11] C. J. D'Orazio dan K. K. R. Choo, "A technique to circumvent SSL/TLS validations on iOS devices," *Futur. Gener. Comput. Syst.*, vol. 74, hal. 366–374, 2017.

[12] S. Fahl, M. Harbach, H. Perl, M. Koetter, dan M. Smith, "Rethinking SSL development in an appified world," *Proc. ACM Conf. Comput. Commun. Secur.*, hal. 49–60, 2013.

[13] S. Gunasekera, *Android Apps Security*. 2020.

[14] V. Moonsamy dan L. Batten, "Mitigating man-in-the-middle attacks on smartphones - A discussion of SSL pinning and DNSSec," *Proc. 12th Aust. Inf. Secur. Manag. Conf. AISM 2014*, hal. 5–13, 2014.

[15] V. Tendulkar dan W. Enck, "An Application Package Configuration Approach to Mitigating Android SSL Vulnerabilities," *MoST*, 2014.

[16] J. Walton, J. Steven, J. Manico, K. Wall, dan R. Iramar, "Certificate and Public Key Pinning," *Open Worldwide Application Security Project*, 2023. .

[17] H. Eland, "Securing a Domain: SSL vs. DNSSEC," 2009. [Daring]. Tersedia pada: https://circleid.com/posts/securing_a_domain_ssl_vs_dnssec. [Diakses: 03-Sep-2023].

[18] A. Mallik, "Man-in-the-middle-attack: Understanding in simple words," *Int. J. Data Netw. Sci.*, vol. 3, no. 2, hal. 77–92, 2019.

[19] B. Bhushan, G. Sahoo, dan A. K. Rai, "Man-in-the-middle attack in wireless and computer networking - A review," *Proc. - 2017 3rd Int. Conf. Adv. Comput. Commun. Autom. (Fall), ICACCA 2017*, vol. 2018-Janua, hal. 1–6, 2018.

[20] R. Jasek, "Security Deficiencies in the architecture and overview of Android and iOS mobile operating systems," *Proc. 10th Int. Conf. Cyber Warf. Secur. ICCWS 2015*, hal. 153–161, 2015.

[21] Apple, "TLS Session Cache," *Apple Developer Forums*, 2015. [Daring]. Tersedia pada: https://developer.apple.com/library/archive/qa/qa1727/_index.html. [Diakses: 01-Jun-2023].