Online-Journals.org

iJIM International Journal of Interactive Mobile Technologies

iJIM | eISSN: 1865-7923 | Vol. 19 No. 10 (2025) | OPEN ACCESS

https://doi.org/10.3991/ijim.v19i10.54017

PAPER

A Biometric Technique to Secure Credit Card Information on Android Devices

Mahmoud Ayyad, Khair Eddin Sabri(✉)

Computer Science Department, King Abdullah II School of Information Technology, The University of Jordan, Amman, Jordan

k.sabri@ju.edu.jo

ABSTRACT

Advances in digital payment systems have increased the need for robust security measures, particularly on Android devices, which are more vulnerable due to their open-source nature. Current applications require authentication, but once users are authenticated, they can proceed with payments if credit card information is stored on the device. This creates a security risk in cases where a mobile device is stolen while an active payment session is active, allowing attackers to make unauthorized payments. This paper addresses the problem of securing sensitive information stored locally on devices. It proposes a technique based on fingerprint and advanced encryption standard (AES) cryptography. Fingerprint is used for authentication, while AES is used for encryption. The technique is implemented as an Android library. The goal is to help developers secure their Android mobile applications through simple APIs without dealing with the complexity of cryptography. Experimental results show that the overhead of integrating the library into applications in terms of time and memory is insignificant.

KEYWORDS

encryption, advanced encryption standard (AES), biometric authentication, Android

1 INTRODUCTION

Android is currently the most popular mobile operating system, with a market share of about 72.04 percent in the fourth quarter of 2024 [1]. Therefore, it has attracted many organizations to provide services through Android mobile applications, such as learning enhancement [2] [3]. Many of these applications handle sensitive data. For example, e-payment applications require credit card information for online shopping. The security of these applications is essential, as the exposure of such information could harm the users. While Android provides robust security features, such as biometrics, their implementation is optional, and no regulations enforce their use in applications on the Google Play Store.

Recent studies [4–8] highlight the spread of mobile payment systems and the need for security measures. Vulnerability in payment systems can lead to serious

Ayyad, M., Sabri, K.E. (2025). A Biometric Technique to Secure Credit Card Information on Android Devices. *International Journal of Interactive Mobile Technologies (iJIM)*, 19(10), pp. 222–235. https://doi.org/10.3991/ijim.v19i10.54017

Article submitted 2024-12-21. Revision uploaded 2025-02-24. Final acceptance 2025-02-25.

consequences such as loss of customer trust and financial loss. While standards like PCI DSS aim to maintain the confidentiality, integrity, and availability (CIA) of the payment system, its implementation is the responsibility of developers who may not be security professionals, which could lead to vulnerabilities. Even when following these standards, vulnerability remains a possibility [5]. With the growth of digital payment systems, ensuring their security is important for protecting both users and organizations.

As Android is the most popular mobile operating system, many organizations are building Android applications, most of which contain sensitive data that should not be accessed by unauthorized users. Additionally, the open-source nature of Android and reduced storage and computational power constraints [9] make it harder for developers to secure their applications and protect user data. The lack of regulations that enforce biometric authentication for payment-related applications could further increase the threats to such applications. Therefore, it is essential to provide developers with security tools and libraries to secure their applications.

This paper focuses on securing sensitive data, such as credit card information, by proposing an Android library that combines biometrics and cryptography. The library is designed to provide developers, especially those without deep security expertise, with a solution for implementing high-level security features in their applications. The contributions of this paper are as follows:

1. Developing a technique that integrates Android's biometrics API and cryptographic techniques to secure sensitive information.
2. Delivering an Android library that abstracts complex security implementations, making it accessible to developers.

The remainder of this paper is organized as follows: Section 2 provides a review of related work. Section 3 details the methodology of the study, followed by results in Section 4. Section 5 presents an analysis and discussion. Finally, Section 6 concludes the study and highlights future research directions.

2 LITERATURE REVIEW

Yong, Christen, and Kruttika [5] examined security issues in mobile payment systems, emphasizing the importance of security regulations such as PCI DSS, which mandates compliance for merchants processing payment cards. They highlighted risks of data breaches, such as fraud and identity theft, and discussed mobile payment security mechanisms like usernames, passwords, multi-factor authentication, and biometric options, such as fingerprint authorization used by Apple Pay and Samsung Pay. Saurav, Aviral, and Pranshu [10] analyzed Android OS security, noting a rise in malicious apps alongside the growing Android user base. They described Android's open-source structure as not inherently detrimental to security and identified risks like reverse engineering, which allows attackers to extract app information. Luminita and Ciprian [11] studied Android fingerprint sensors, detailing how the system processes fingerprints during enrollment and authentication. They discussed vulnerabilities such as dummy fingerprints, which exploit residual fingerprints left on surfaces, raising questions about their security compared to traditional passwords. Xinman, Tingting, and Xuebin [12] presented a survey about Android-based biometric verification systems, stating that fingerprints are the most common due to their affordability and maturity. They evaluated biometric performance using False Reject Rate (FRR), False Accept Rate (FAR), and Equal Error Rate (EER). The paper

concludes that biometrics are becoming a viable alternative to traditional cryptographic methods. Iram and Yasir [13] proposed a biometric-based authentication system for Android applications, highlighting its reliability, scalability, and security. They categorized fingerprint-matching techniques and emphasized technologies like classification, indexing, and cosmic pattern analysis for accuracy. Their system improved session timeout management through adaptive fingerprint authentication, incorporating encryption to protect biometric data and enhance user privacy.

Saranya and Naresh [14] introduced mobile security for e-healthcare applications using key distribution. They proposed a Secure Authentication Protocol (SAP) for mobile payments in e-healthcare applications to ensure data user account safety and confidentiality. The protocol uses cryptographic techniques to provide authentication between the client and server. They conducted a performance analysis that shows that the protocol achieves security with lower computational costs compared to existing methods. Parandhaman [15] provides a lightweight and privacy-preserving security mechanism for mobile payment within the context of the IoT. First, it identified vulnerabilities in the underlying Certificate-less Signature System (CSS). Then, it proposed an improved CSS structure that is formally verified using the Forking Lemma. Furthermore, the paper introduced a secure transaction protocol for Android mobile payment systems. The paper of [16] studies the payment credential leaks for four top-tiered cashiers that serve over one billion users and tens of millions of merchants globally. The research identifies new sources of credential leaks, uncovering four types of exploits resulting from these leaks and additional implementation flaws. To address this, the authors developed an automated tool, PayKeyMiner, which detected approximately 20,000 leaked payment credentials across thousands of apps. The findings were reported to the affected cashiers, who acknowledged the issues and pledged to notify the affected merchant apps. Some apps have since updated their leaked credentials to mitigate the risks.

Other researchers used the advanced encryption standard (AES) cipher in their Android applications to provide confidentiality, such as [17] [18]. In [17], the paper introduced a mobile application called AMEACrypt. The application has many features, such as a password generator, password vulnerability checker, and file encryption and decryption functions. The goal is to ensure user security through stronger passwords or data encryption using the AES cipher. In [18], end-to-end encryption is introduced to secure the communication of instant messages. The encryption and decryption of messages is based on the AES cipher.

This literature review covered papers related to mobile payment applications, fingerprint authentication, and AES encryption. As a result, we conclude on the importance of integrating biometric authentication to be used for accessing sensitive data such as credit card information in payment systems. In many applications, biometrics is used for login authentication into the application, but once logged in, the user can access all sensitive data inside the application. If the mobile is stolen while the application's session is still active, then there is a risk of the exposure of sensitive data. Therefore, implementing a library that can be used to secure sensitive data by requiring biometric authentication even if the user is logged in is important and can help developers protect sensitive user data in their applications.

3 METHODOLOGY

This section introduces the proposed technique, which is implemented as an Android library. The library is based on the Android Biometric API and AES for authentication and data encryption.

3.1 Android Keystore system

The Android Keystore system is used in the proposed technique to securely store the AES encryption key. The Keystore allows cryptographic keys to be stored securely and makes them accessible to authorized users only. Authentication, such as fingerprint, can restrict access to these keys. The Keystore system ensures cryptographic keys are not revealed to unauthorized users even if the application is compromised. They are stored in a secure environment, such as the Trusted Execution Environment (TEE) [19] or Secure Element (SE), to ensure the security of these keys even if the device's internal storage is breached.

Since Android 9 (API level 28), encrypted keys can be securely stored in the Keystore system without revealing their content in plaintext. Biometric authentication ensures that only authorized users can access the keys, making fingerprint-based authentication particularly effective. This method eliminates human error, such as forgetting passwords. The Keystore system provides facilities in the proposed technique to restrict when and how keys can be used, such as requiring user authentication through fingerprint for each access to the AES cipher key in a similar flow to what is shown in Figure 1.

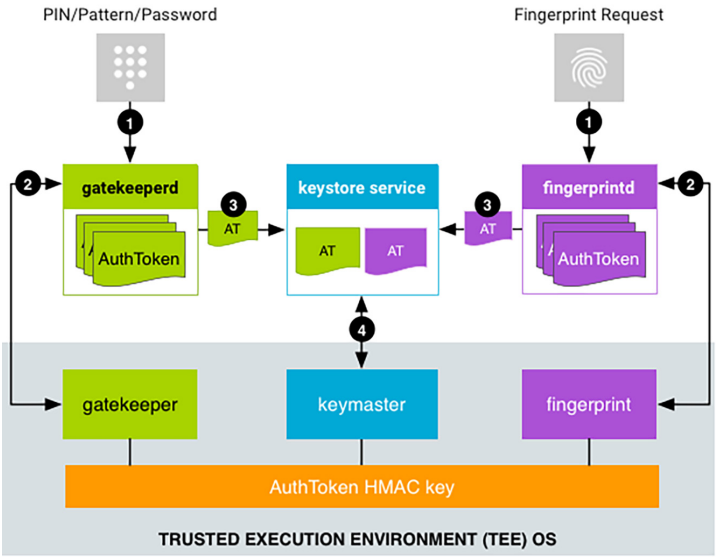

Fig. 1. Getting a key from the Keystore system using authentication [20]

3.2 The proposed technique

The technique aims to restrict access to sensitive data and be accessible only to authorized users through authentication using fingerprints. The biometric authentication is used to secure AES encryption keys stored in the Android Keystore.

Generate cryptographic keys. The first step is to generate an AES cryptographic key using the Android Keystore. The key is configured with specific parameters, such as alias, encryption, and decryption purposes, and AES-256 encryption with PKCS7 padding. Fingerprint authentication is required to access the key. Once configured, the key is generated and stored securely in the Keystore. The corresponding cipher instance is initialized for encryption or decryption, incorporating an Initialization Vector (IV) to prevent data repetition and ensure unique outputs. The IV can be stored alongside the encrypted data since it does not compromise security.

Algorithm 1: Pseudo Code for Key Generation and Cipher Creation

1. CREATE KeyGenParameterSpec object for key configuration with the following settings:
   - SET Alias for the key.
   - SET Purpose to ENCRYPT.
   - SET Key size to 256 bits.
   - SET Encryption padding to PKCS7 padding.
   - SET User authentication to Required.
2. CREATE KeyGenerator object for the key generator based on the key configuration with the following setting:
   - SET Algorithm to AES.
   - SET Provider to AndroidKeyStore.
3. CALL generateKey() method using the key generator object to generate a key
4. CREATE encryption cipher object with the following settings:
   - SET Algorithm to AES/CBC/PKCS7Padding.
   - SET Mode to ENCRYPT_MODE.
   - SET Key to the generated key.
5. CREATE decryption cipher object with the following settings:
   - SET Algorithm to AES/CBC/PKCS7Padding.
   - SET Mode to DECRYPT_MODE.
   - SET Key to the same generated key.
   - SET initialization vector (IV)

Check Android device support. Not all Android devices support biometric authentication. Developers can verify device compatibility using the 'BiometricManager.from(context).canAuthenticate()' function, which returns one of the following values:

- BIOMETRIC_ERROR_HW_UNAVAILABLE
- BIOMETRIC_ERROR_NONE_ENROLLED
- BIOMETRIC_ERROR_NO_HARDWARE
- BIOMETRIC_SUCCESS

Biometric authentication is only enabled on devices that meet Android's security standards, such as fingerprint matching in a trusted execution environment.

Biometric prompt. A biometric prompt is used to authenticate users and access the encryption or decryption ciphers. The prompt is built with parameters such as UI context, an executor for callback events, and an authentication callback. Upon successful authentication, the cipher is made available for use.

Algorithm 2: Pseudo Code for Biometric Prompt

1. CREATE BiometricPrompt object as
   biometricPrompt = BiometricPrompt(fragmentContext, mainExecutor, callBack)
2. Implement the AuthenticationCallback method to handle the results of the biometric authentication as
   IF successful THEN retrieve the cipher
   ELSE IF authentication_error THEN log_error_message
   ELSE IF authentication_failed THEN log_failed_message
3. Build Prompt Configuration such that
   SET title to "Authentication Required"
   SET navigateButtonText to "Cancel"
4. CALL biometricPrompt.authenticate() with the built promptConfig and pass a CryptoObject containing the cipher (used for encryption/decryption) to authenticate users.

The encryption and decryption ciphers remain unusable until the user is successfully authenticated via the biometric prompt. Figure 2 shows the workflow of

the proposed technique, which can be summarized as follows. The technique starts with setting the configuration of the cryptographic keys by setting parameters such as key size, padding, algorithm, etc. Then, it checks if the device supports biometric authentication. In case it has no support, the technique terminates, and another way of authentication should be used. If supported, both encryption and decryption ciphers are created, and their keys are stored in the Keystore. In order to access these ciphers, biometric authentication is required, and the user should be authenticated.

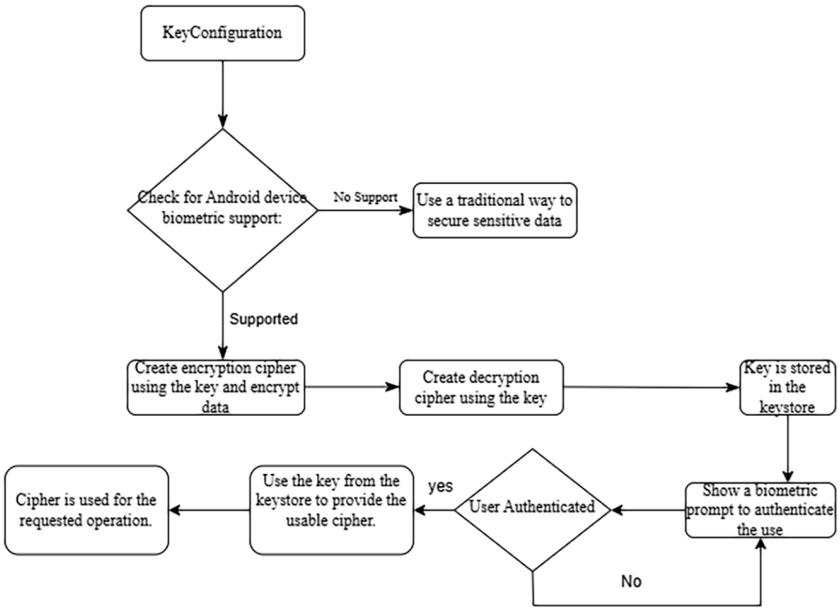

Fig. 2. Implementation flowchart

3.3 The library and integration guideline

The proposed technique is packaged as an open-source Android library to help organizations secure sensitive data without requiring deep cryptographic knowledge. The proposed library is implemented in Kotlin and is designed to work with devices supporting Android 9 and above. The implementation of the library includes key generation and storage, as shown in Algorithm 1. Then, the biometric fingerprint is used for authentication as presented in Algorithm 2. After authentication, the library provides methods to encrypt and decrypt data, using AES in CBC mode with PKCS7 padding.

Developers can integrate the library with their applications by creating a KeyManager instance and then calling simple functions such as encryptData() and authenticateUser(). To clarify, the following steps should be followed. First, the developer should add the library dependency to the Android project. Then, initialize the KeyManager class with the required context and listener.

```
val keyManager = KeyManager(fragmentActivity, listener)
```

The function encryptData(String keyAlias, String data) is used to securely encrypt sensitive data using a generated AES key.

```
keyManager.encryptData(keyAlias, data)
```

Users are authenticated via authenticateUser() before decrypting sensitive data. The biometric prompt is a built-in support for user authentication with clear callbacks for success or failure.

```
keyManager.authenticateUser()
```

The function decryptData(String keyAlias, byte[] encryptedData) is used to decrypt data after user authentication.

```
keyManager.decryptData(keyAlias, encryptedData)
```

Figure 3 shows an example of the biometric prompt shown to the user when authenticate user is called using the library. The library's source code and integration instructions are available on GitHub at https://github.com/Mahmoud-Mustafa-Dev/AESBio/. However, the library is currently a prototype and is not recommended for production use.

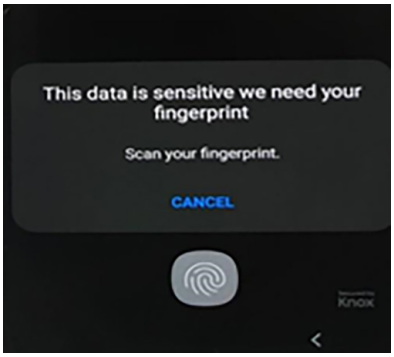

Fig. 3. Biometric prompt

In case of authentication failures, the library uses Android's built-in biometric error handling to manage the failures. The Android biometric API provides detailed error messages for scenarios such as hardware unavailability, no enrolled biometrics, and authentication failure. The library communicates these error messages directly to developers. The library does not implement additional error-handling mechanisms other than the one provided by the Android biometric system.

4 EXPERIMENTAL RESULTS

This section outlines the development environment and tools used to implement the library, as well as the experimental setup used to evaluate its performance. The library was developed in Kotlin using Android Studio as the integrated development environment (IDE). GitHub was used for version control, and the library was hosted on jetpack.io. To evaluate the library, an empty Android application was built and deployed on two mobile devices: Galaxy A51 and Galaxy A70 models. The library was integrated into the empty Android project to assess the introduced overhead in terms of execution time, memory consumption, and energy used. The Android Profiler was used for measurement.

Galaxy A51: First, we created an empty Android project without integrating any external libraries. The memory usage for the empty application was 96.7 MB, while it did not show any CPU activity or energy consumption. Then, we integrated the

library into the empty project and re-ran the project. The memory usage increased slightly from 96.7 MB to 97.1 MB. CPU and energy indicators remained unchanged. Finally, we performed encryption and decryption operations within the application to evaluate their impact on performance. The memory usage increased to 103.4 MB, while CPU and energy remained stable.

Galaxy A70: To account for variations in device specifications, the same experiment was conducted on the Galaxy A70 model. Similar to the A51, the empty app showed no activity in the CPU and energy indicators. However, the memory usage for the empty application was higher at 136.3 MB, reflecting differences in how devices allocate resources to native operations, graphics, and other background tasks. After library integration, memory usage increased from 136.3 MB to 138.1 MB, with no changes in CPU or energy indicators. During encryption and decryption, memory usage increased to 140.5 MB, while CPU and energy indicators remained stable.

The execution time for the applications was measured on both devices, and the results are summarized in Table 1. The results indicate that the library adds minimal overhead to application performance, with only a negligible increase in execution time.

Table 1. Experiment execution time

Model	Empty App Execution Time	Integrated App Execution Time
A51	1.03 seconds	1.04 seconds
A70	710 milliseconds	740 milliseconds

The library functioned as expected during testing. Upon submitting sensitive data, the application prompted the user to authenticate using their fingerprint to access the encryption cipher. Similarly, when encrypted data was required (e.g., during a digital payment), the user was prompted to authenticate again to decrypt the data. This ensures compliance with the PCI DSS control objective for protecting cardholder data.

5 ANALYSIS AND DISCUSSION

To evaluate the added value of integrating the proposed technique into the digital payment process, it is essential to first examine the traditional approach to initiating digital payments. Typically, organizations do not require additional authentication for purchases, as users are already authenticated during the login phase. Login sessions are commonly used to confirm recent authentication. However, this practice introduces a significant security risk: if a device falls into the wrong hands while the user is still logged in, an attacker can easily perform unauthorized transactions, especially when payment card information is saved on the device. For instance, if a phone is stolen and the attacker gains access to applications where the user remains logged in, they could make payments or conduct other sensitive transactions. Although organizations generally provide mechanisms for users to report lost devices, these risks could be mitigated by requiring fingerprint authentication before processing any digital payment. Some applications require two-factor authentication to complete a payment; even this technique is not secure in some scenarios. Two-factor authentication is usually implemented as a code sent to the mobile or to the user's email. If an unlocked device falls into the hands of an attacker, then the attacker could make payments or complete money transfers, as the user's email or other authentication methods are usually accessible on the compromised device.

By requiring fingerprint authentication before making payments, these vulnerabilities could be effectively addressed.

While cryptographic methods are effective for securely storing sensitive data, they do not resolve the issue of verifying the legitimate owner of a device once it has been compromised. This highlights the crucial role of authentication in ensuring security. However, authentication methods must be designed in a way that does not hinder user experience. Fingerprint recognition was chosen for its availability, ease of use, and reliability on modern Android devices. Unlike passwords or PINs, biometrics offer convenience, as there is no need to remember passwords or codes; security, as they are resistant to phishing, brute force, and social engineering attacks; and finally, quick authentication. Alternative methods such as facial recognition and voice authentication were avoided due to higher error rates and inconsistent hardware support across Android devices. They provide a layer of security that is both convenient and robust, making them an ideal solution for enhancing the security of digital payment systems.

5.1 Performance analysis

In this section, we analyze the performance results obtained from the previous section. First, we analyze the execution time and then the memory usage. In the previous section, the integration overhead was measured to assess the impact of the library on the execution time on both A51 and A70 devices. In A51, the execution time is increased by 0.01 seconds, which is equivalent to 0.97%. While on A70, the execution time is increased by 30 milliseconds, which is equivalent to 4.23%. These results indicate that the overhead introduced by the library is negligible on both devices. The analysis shows that the execution time on A70 is less than on A51 by 29%. This is because of the higher hardware specification of A70 over A51.

Regarding memory usage, the experiment shows that after integration, the memory usage is increased by 0.4 MB on the A51 and by 1.8 MB on the A70. For encryption and decryption, the memory usage is increased by 6.7 MB (6.48%) on the A51 and by 4.2 MB (3.08%) on the A70. It is of note that the memory usage for the Galaxy A70 was higher than the A51. Based on that, we can conclude that the library is lightweight, as the overhead produced by integrating the library is not significant.

5.2 Security analysis

The proposed approach combines biometric authentication and AES encryption to secure sensitive data effectively. The security of the library is based on the assumption that the TEE in Android and the AES encryption are secure. If the library is properly used, then different types of attacks are not applicable; for example:

- Impersonating attacks is not applicable as fingerprints provide a unique identifier, making it difficult for attackers to impersonate legitimate users. Furthermore, advanced biometric systems incorporate liveness detection techniques, which may detect fake fingerprints.
- Sensitive data is encrypted through AES encryption, which is known for its security against different attacks such as brute-force attacks, known plain-text attacks, etc., especially when an IV is used and a secure encryption mode is used, such as CBC mode with PKCS7 padding.

- The library requires that biometric authentication take place in a TEE or using SE hardware. These environments ensure that the biometric template and authentication process are isolated and tamper-resistant, preventing spoofing at the software level.
- The Android Keystore ensures that biometric data and authentication keys are encrypted and cannot be exported. Even if an attacker gains root access to the device, they cannot retrieve the raw biometric data or use it to spoof authentication.
- Session hijacking is not applicable in our case, as the proposed technique does not depend on sessions to access sensitive data such as credit card information; i.e., every access requires fingerprint authentication.
- Injection Attacks such as SQLi are not applicable, as the only input from the user is the fingerprint, which is checked through the Android API.

Advanced encryption standard. Advanced encryption standard [21] was selected over alternatives like DES for its security and efficiency, in addition to being identified as the symmetric key encryption standard. For instance, the key size used in the proposed library is 256 compared to 64 bits in DES, and the block size is 128 bits compared to 64 bits in DES. Furthermore, DES was deprecated by NIST in 2005 due to vulnerabilities [22]. Other alternative methods could be the public key algorithm, which is not used as it is inefficient for large data due to high computational overhead; 3DES for performance issues compared to AES [23]; and Password-Based Encryption (PBE), as it is prone to human error and weak passwords.

Biometric spoofing. While fingerprint provides a secure way of user authentication, it is not entirely immune to threats such as spoofing attacks. Biometric spoofing refers to the act of creating a counterfeit biometric sample to deceive the authentication system. In the case of fingerprint authentication, attackers might use fake fingerprints made from materials such as silicone, gelatin, or other synthetic substances. These replicas can sometimes deceive fingerprint scanners, especially older or less sophisticated hardware. However, modern Android devices employ liveness detection, which ensures that the fingerprint being scanned belongs to a live human rather than a dummy or replica. Features such as pulse detection and skin conductivity can enhance the scanner's ability to detect spoofing attempts. Furthermore, several techniques are proposed in the literature for fingerprint spoofing detection [24].

5.3 Analysis of failure scenarios and fallback mechanisms

There could be failure scenarios caused by different reasons. For example, biometric authentication failure could be caused by issues such as wet fingers, sensor damage, or poor enrollment. Another reason could be using devices that do not support biometric hardware. As a fallback mechanism, the BiometricManager API is used to verify hardware such that if biometrics is not supported by a device, it falls back to password-based encryption. Also, clear error messages are provided to developers in case of failure.

5.4 Comparison with existing solutions

The proposed solution differentiates itself from existing biometric and cryptographic implementations by offering a lightweight, developer-friendly library tailored for Android devices. A brief comparison with baseline approaches is shown in Table 2.

Table 2. Comparison with existing solutions

Feature	Proposed Solution	Comparison with Other Solutions
Integration Complexity	Simplifies integration with minimal setup, suitable for non-experts.	Often requires deep expertise in cryptography or custom implementations.
Security	Combines AES encryption with biometric authentication, ensuring compliance with PCI DSS.	Some solutions use password-based encryption, which is more prone to user error and brute force.
Performance	Negligible impact on memory and execution time, as demonstrated in tests.	Performance can vary; some libraries add significant overhead.
Hardware Dependency	Utilizes Android's TEE/SE for enhanced security.	Not all solutions enforce hardware-backed key storage, increasing vulnerability to attacks.
Biometric Integration	Fully integrates with Android's Biometric API for seamless user authentication.	Limited or partial support for biometric features on Android.

Unlike generic solutions, this library is specifically designed to address the security challenges of storing sensitive payment information on Android devices. It bridges the gap between usability and advanced security features, making it ideal for startups and developers with limited security expertise.

5.5 User experience

There are two types of users we are approaching: developers and end-users. The primary target for our system is Android developers, as the main goal is to provide security through a library that is easy to use and integrate. As presented in the implementation section, only a few APIs are required. As a proof of concept of the usability of the library, the library was provided to 30 fourth-year university students to integrate with their mobile applications. All the students are enrolled in one of the information technology majors and have prior knowledge of Android mobile applications. As a result, 90% of the students were able to use the library and integrate it into their applications without help, while the rest required support for the integration. For end users, the library provides an easy way to authenticate them based on fingerprint, which is a user-friendly method and is widely used in many applications, such as, for example, to record attendance [25].

5.6 Real-world applicability

The proposed technique and library are applicable to any modern Android applications handling sensitive data such as e-commerce, banking, and digital wallets. For example, the library can be used by e-commerce applications to enhance security for payment transactions by requiring biometric authentication before accessing stored credit card details. Furthermore, it can be used by banking applications to secure sensitive account information, ensuring only authorized users can perform actions like fund transfers or bill payments. Finally, in digital wallets, the library can be used to protect stored payment methods.

By combining biometric authentication with encryption, the library addresses real-world challenges such as stolen devices and unauthorized access and ensures compliance with security standards such as PCI DSS. The library is lightweight, which does not affect the performance of the applications.

6 CONCLUSION AND FUTURE WORK

In this paper, we introduce a technique that is based on biometrics and cryptography to secure sensitive data on Android devices. The technique is implemented as a library that can be used by Android developers to integrate with their applications, including those without in-depth knowledge of security best practices on Android. The proposed library uses the Android biometrics API and the AES encryption algorithm to protect sensitive data, ensuring that only authenticated users can access it. The encryption and decryption keys are securely stored in the Android keystore system and are only accessible after successful fingerprint authentication.

An experiment was conducted to evaluate the impact of integrating the library with Android applications. The results show that the library does not significantly affect execution time and memory usage. The execution time is increased by 0.97% on the A51, while on the A70, the execution time is increased by 4.23%. The memory usage is increased by 6.48% on the A51 and by 3.08% on the A70. A theoretical analysis showed that combining fingerprint authentication with cryptography provides a robust security layer. By requiring fingerprint authentication to access sensitive data such as credit card information, it prevents an attacker from making payment even in the case of the existence of a valid session. Therefore, it mitigates risks such as session hijacking, root access attacks, and losing passwords.

Given its lightweight implementation and the simplicity of integration through APIs, the library is an excellent choice for developers looking to secure sensitive data without introducing significant computational overhead. By combining biometric authentication with cryptography, the library provides an efficient way of protecting sensitive data on Android devices.

We believe there should be stricter security regulations from the Google Play Store. As the platform processes numerous application submissions daily, an approach should be implemented to ensure that no harmful applications are made available to the public. Specifically for digital payment applications, it would be ideal if the Google Play Store required biometric authentication as a prerequisite for digital payments. Applications failing to meet this requirement should be rejected, with clear feedback on the reason for rejection provided to developers. This measure would enhance the security and reliability of digital payment systems. Furthermore, currently, Android users can enroll multiple fingerprints; however, during the matching phase, all enrolled fingerprints are treated equally. The fingerprint matcher simply checks if the input fingerprint matches any of the enrolled fingerprints, without distinguishing between them. There should be a way to enable assigning different privileges to different fingerprints. For example, in the case of a shared family tablet, the father's fingerprint could allow purchases from all applications, while the daughter's fingerprint could permit purchases from a limited selection of applications. This functionality would provide greater flexibility and security for multi-user devices.

7 REFERENCES

[1] Ahmed Sherif, "Android – Statistics & Facts," *Statista*, 2024. [Online]. Available: https://www.statista.com/topics/876/android/

[2] H. H. Batubara, M. S. Sumantri, and A. Marini, "Developing an Android-based E-Textbook to improve learning media course outcomes," *International Journal of Interactive Mobile Technologies*, vol. 16, no. 17, pp. 4–19, 2022. https://doi.org/10.3991/ijim.v16i17.33137

[3] Asmianto, M. Hafiizh, D. Rahmadani, K. Pusawidjayanti, and S. Wahyuningsih, "Developing Android-based interactive e-modules on trigonometry to enhance the learning motivation of students," *International Journal of Interactive Mobile Technologies*, vol. 16, no. 2, pp. 159–170, 2022. https://doi.org/10.3991/ijim.v16i02.27503

[4] S. Bojjagani, V. N. Sastry, C.-M. Chen, S. Kumari, and M. K. Khan, "Systematic survey of mobile payments, protocols, and security infrastructure," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, pp. 609–654, 2023. https://doi.org/10.1007/s12652-021-03316-4

[5] Y. Wang, C. Hahn, and K. Sutrave, "Mobile payment security, threats, and challenges," in *Proc. 2nd Int. Conf. Mobile Secure Services (MobiSecServ)*, 2016, pp. 1–5. https://doi.org/10.1109/MOBISECSERV.2016.7440226

[6] A. Mos and M. M. Chowdhury, "Mobile security: A look into Android," in *IEEE International Conference on Electro Information Technology (EIT)*, 2020, pp. 638–642. https://doi.org/10.1109/EIT48999.2020.9208339

[7] A. Sharma, S. Singh, S. Kumar, A. Chhabra, and S. Gupta, "Security of Android banking mobile Apps: Challenges and opportunities," in *International Conference on Cyber Security, Privacy and Networking (ICSPN 2022)*, in Lecture Notes in Networks and Systems, N. Nedjah, G. Martínez Pérez, and B. B. Gupta, Eds., vol. 599, Springer, Cham, 2023, pp. 406–416. https://doi.org/10.1007/978-3-031-22018-0_39

[8] S. Alshebli and C. Y. Yeun, "Examining the security landscape of mobile payment systems," in *2024 2nd International Conference on Cyber Resilience (ICCR)*, Dubai, United Arab Emirates, 2024, pp. 1–6. https://doi.org/10.1109/ICCR61006.2024.10532814

[9] W. Ahmed *et al.*, "Security in next generation mobile payment systems: A comprehensive survey," *IEEE Access*, vol. 9, pp. 115932–115950, 2021. https://doi.org/10.1109/ACCESS.2021.3105450

[10] S. Yadav, A. Apurva, P. Ranakoti, S. Tomer, and N. R. Roy, "Android vulnerabilities and security," in *2017 Int. Conf. Comput. Commun. Technol. Smart Nation (IC3TSN)*, 2017, pp. 204–208. https://doi.org/10.1109/IC3TSN.2017.8284477

[11] L. Apostol and C. Dobre, "Android fingerprint sensor: Pitfalls and challenges," in *2018 IEEE 16th Int. Conf. Embedded Ubiquitous Comput. (EUC)*, 2018, pp. 133–137. https://doi.org/10.1109/EUC.2018.00027

[12] X. Zhang, T. He, and X. Xu, "Android-based smartphone authentication system using biometric techniques: A review," in *2019 4th Int. Conf. Control, Robotics Cybern. (CRC)*, 2019, pp. 104–108. https://doi.org/10.1109/CRC.2019.00029

[13] I. T. E. J. Mohammed and Y. A. Mohamed, "A new system for user authentication using android application," in *2020 Int. Conf. Comput., Control, Electr., Electron. Eng. (ICCCEEE)*, 2021, pp. 1–5. https://doi.org/10.1109/ICCCEEE49695.2021.9429637

[14] A. Saranya and R. Naresh, "Efficient mobile security for e-health care application in cloud for secure payment using key distribution," *Neural Processing Letters*, vol. 55, pp. 141–152, 2023. https://doi.org/10.1007/s11063-021-10482-1

[15] V. P. Parandhaman, "A secured mobile payment transaction handling system using Internet of Things with novel cipher policies," in *2023 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, Chennai, India, 2023, pp. 1–8. https://doi.org/10.1109/ACCAI58221.2023.10200255

[16] S. Shi, X. Wang, K. Zeng, R. Yang, and W. C. Lau, "An empirical study on mobile payment credential leaks and their exploits," in *Security and Privacy in Communication Networks (SecureComm 2021)*, in Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, J. Garcia-Alfaro, S. Li, R. Poovendran, H. Debar, and M. Yung, Eds., vol. 399, Springer, Cham, 2021, pp. 79–98. https://doi.org/10.1007/978-3-030-90022-9_5

[17] J. E. Borromeo, R. A. Pagaduan, J. E. Avestro, R. C. B. Pinca, J. N. Garcia, and Z. D. Gremio, "AMEACrypt: Android mobile encryption application generating password using AES Algorithm," in *Proceedings of Ninth International Congress on Information and Communication Technology ICICT,* X. S. Yang, S. Sherratt, N. Dey, and A. Joshi, Eds., vol. 1000, Springer, Singapore, 2024, pp. 455–467. https://doi.org/10.1007/978-981-97-3289-0_37

[18] Nurhayati, Kastari, and F. Fahrianto, "End-To-End encryption on the instant messaging application based android using AES cryptography algorithm to a text message," in *2022 10th International Conference on Cyber and IT Service Management (CITSM)*, 2022, pp. 1–6. https://doi.org/10.1109/CITSM56380.2022.9935963

[19] C. Shepherd and K. Markantonakis, *Trusted Execution Environments*. Cham: Springer, 2024. https://doi.org/10.1007/978-3-031-55561-9

[20] Android Developers, "Platform Architecture," 2021. [Online]. Available: https://developer.android.com/guide/platform

[21] National Institute of Standards and Technology-Federal Inf. Process. Stds. (NIST FIPS), "Advanced Encryption Standard (AES)," 2023. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=936594 [Accessed: Feb. 2025].

[22] National Institute of Standards and Technology, "NIST withdraws outdated data encryption standard," 2005. [Online]. Available at: https://www.nist.gov/news-events/news/2005/06/nist-withdraws-outdated-data-encryption-standard [Accessed: Feb. 2025].

[23] H. Dibas and K. E. Sabri, "A comprehensive performance empirical study of the symmetric algorithms: AES, 3DES, Blowfish and Twofish," in *2021 International Conference on Information Technology (ICIT)*, 2021, pp. 344–349. https://doi.org/10.1109/ICIT52682.2021.9491644

[24] C. Yuan, Z. Xu, X. Li, Z. Zhou, J. Huang, and P. Guo, "An interpretable siamese attention Res-CNN for fingerprint spoofing detection," *IET Biometrics*, vol. 2024, p. 6630173, 2024. https://doi.org/10.1049/2024/6630173

[25] B. Utomo, A. T. Wibowo, M. Ridwan, M. A. Izzuddin, A. B. Gumelar, and S. Arifin, "Enhanced of attendance records technology used geospatial retrieval based on crossing number," *International Journal of Interactive Mobile Technologies*, vol. 14, no. 16, pp. 101–116, 2020. https://doi.org/10.3991/ijim.v14i16.13911

8 AUTHORS

Mahmoud Ayyad is a Senior Android Engineer at Careem with a Master's degree in Computer Science. He began his career in 2017 as a mobile game developer and then shifted to Android development in 2020. Since then he has worked with companies such as Sociumtech, Almosafer, and most recently Careem. Currently, he is planning to expand his research and pursue a PhD.

Khair Eddin Sabri is a Professor of Cybersecurity at the University of Jordan. He earned his Ph.D. in Software Security from McMaster University, Canada, in 2010 and his M.Sc. in Computer Science from the University of Jordan in 2004. His research focuses on software security, malware analysis, web application security, formal analysis of security properties, access control, intrusion detection systems, and blockchain and smart contracts (E-mail: k.sabri@ju.edu.jo).