



SECURITY ANALYSIS OF TWO-FACTOR AUTHENTICATION APPLICATIONS: VULNERABILITIES IN DATA STORAGE AND MANAGEMENT

¹Dzikri Izzatul Haq, ²Syafrial Fachri Pane, ³M. Amran Hakim Siregar

^{1,2}Applied Bachelor Program of Informatics Engineering, Bandung, Indonesia

³Computer Science, Information Technology and Actuarial Science, Jakarta, Indonesia

¹1214062@std.ulbi.ac.id, ²*syafrial.fachri@ulbi.ac.id, ³amranhakimsiregar@umiba.ac.id

*correspondence email : syafrial.fachri@ulbi.ac.id

Abstract

In the digital era, two-factor authentication (2FA) is used as an additional security measure to protect user access to digital services. However, the storage methods of authentication data in 2FA applications have potential security vulnerabilities that can be exploited. This study analyzes five popular 2FA applications, namely Google Authenticator, 2FAS, Aegis Authenticator, Okta Verify, and TOTP Authenticator, focusing on how secret keys are stored and the potential exploitation risks. The experiment was conducted in a virtual Android environment using rooted BlueStacks 5. Data acquisition was performed using Media Manager and X-Plore File Manager, followed by data analysis with SQLite Browser and bypass with PyOTP. The results showed that two out of five applications stored secret keys in plaintext, while others implemented varying degrees of encryption. Exploitation was demonstrated by replicating OTP codes using the PyOTP library based on extracted keys. These findings highlight that improper data storage practices can lead to authentication bypass, posing risks to millions of users. These vulnerabilities may significantly impact user trust and highlight the urgent need for application developers to improve data storage security standards. This research emphasizes the necessity for developers to employ secure key storage mechanisms such as hardware-backed encryption (Android Keystore) to mitigate vulnerabilities, underscoring broader implications for application security.

Keywords: Application Security, Security Analysis, Encryption, OTP Bypass, Two-Factor Authentication

INTRODUCTION

In today's digital era, authentication has become an important element in protecting user data from increasingly complex cyber threats. One widely adopted method is Two-Factor Authentication (2FA), which combines two types of verification: something the user knows, such as a password, and something the user has, such as a token or code. This method provides an additional layer of protection against threats such as phishing and credential theft, which are increasingly prevalent in cyberspace. Research shows that although 2FA is considered more secure than traditional authentication methods, there are still weaknesses that need to be evaluated, especially related to the management and storage of authentication data [1][2].

One of the main drawbacks of 2FA is the way apps store sensitive data, such as secret keys and verification codes. Some 2FA apps, such as Google Authenticator and Okta Verify, have demonstrated better storage practices, using strong encryption to protect sensitive data [3]. However, there are also apps that store data in plaintext format or with weak encryption, making them vulnerable to unauthorized access [4][5]. This study aims to analyze data security vulnerabilities in 2FA applications and compare five popular applications: Google Authenticator, 2FAS Authenticator, Aegis Authenticator, Okta Verify, and TOTP Authenticator. This

technology for data security is very important because it aims to meet the business process needs of users or organizations of a company [6].

The analysis was conducted using a security approach using a rooted virtual Android platform, namely BlueStacks 5. This method allows access to the application's internal folders without the risk of damaging the physical device. The extracted data includes files such as databases, XML, and JSON, which often store important information such as account names, issuers, and secret keys [7][8]. The analysis results show that some applications have significant weaknesses in terms of data storage, which can be exploited by third parties. This indicates that although 2FA is improving security, its implementation still needs improvement to protect users from increasingly sophisticated cyber threats [1][9]. Comparisons between applications show variations in the level of security. For example, applications such as Okta Verify have better data storage practices, using strong encryption and additional protection mechanisms [3][4]. On the other hand, applications such as Aegis Authenticator show significant weaknesses, where sensitive data can be easily extracted from the application's internal folders [2][5]. This study emphasizes the need to strengthen data storage mechanisms in 2FA applications, such as using stronger encryption and secure data management practices [10].

In addition, this study also provides recommendations for users to choose a more reliable 2FA application based on the results of this evaluation. Users are advised to prioritize applications that have a good reputation for security and that actively update their data storage practices to protect sensitive information [11]. Thus, while 2FA is a step forward in improving digital security, it is important for users to remain vigilant and choose applications that offer adequate protection against cyber threats [1][2]. In a broader context, it is important to understand that digital security does not only depend on the technology used but also on the awareness and actions of users. Users need to be trained to recognize potential threats and understand how to protect their information, including the effective use of 2FA [2][12]. With this holistic approach, it is hoped that a safer and more protected digital environment can be created from various existing cyber threats [1][9].

Given the rapid development of technology, it is also important to consider new innovations in the field of authentication. For example, technologies such as Google Passkeys offer a new approach that can replace traditional authentication methods in a more secure and efficient manner [13]. Further research is needed to explore the potential of these technologies in improving authentication security and protecting user data from evolving threats [7][14]. One of the challenges faced by 2FA is its vulnerability to increasingly sophisticated phishing attacks. Although 2FA is designed to provide an additional layer of security, well-designed phishing attacks can exploit this weakness [12][15]. Therefore, it is important for users to not only rely on 2FA but also to develop good security habits, such as verifying the authenticity of a website before entering sensitive information [2][12].

In this study, it was also found that although there are many 2FA applications available, not all applications have the same level of security. Some applications may have additional features that increase security, such as biometric recognition or tighter security settings, while others may not have such features [3][4]. Therefore, users should do careful research before selecting a 2FA application to ensure that they are using the most secure and effective solution [11]. In addition, it is important to consider the impact of organizational policies and practices in implementing 2FA. Organizations should ensure that they not only implement 2FA but also provide training to employees on how to use this system safely and effectively [9]. By increasing awareness and understanding of 2FA, organizations can reduce the risks associated with cyberattacks and better protect their sensitive data [2][9].

While Two-Factor Authentication (2FA) is a step forward in improving digital security, there are still many challenges that need to be overcome. Users and organizations must work together to ensure that good security practices are implemented and that the technology used for authentication is continuously updated and improved [1][9]. With a comprehensive approach, it is hoped that digital security can be improved and cyber threats can be minimized [16][2].

The results of this study are also very important because they provide insight into the security level of 2FA applications used by millions of users worldwide. By exploring the possibility of bypassing authentication using the discovered secret key, this study also highlights the importance of more secure key management and fast and effective security updates and patches. In addition, the findings in this study can be a reference for application developers in improving the security of storing user authentication data.

Prior works, such as Berrios et al. (2023)[17], conducted broad forensic studies across multiple platforms, identifying plaintext or weakly encrypted secret keys. However, these studies primarily cataloged artifacts without extensive practical demonstrations of potential exploitation.

This study specifically addresses this gap by focusing on practical exploitability, using security analysis methods in a controlled environment to clearly demonstrate TOTP bypasses on popular Android-based 2FA apps. The main objectives are to:

1. Analyze storage vulnerabilities in five popular Android 2FA applications.
2. Demonstrate practical OTP bypass methods.
3. Provide clear security recommendations to strengthen authentication data protection.

These contributions are expected to enhance practical security standards for both application developers and digital investigators.

METHODS

The research approach involves using BlueStacks 5 as a rooted virtual Android to create a secure testing environment. Starting from creating and testing scenarios first by setting up five 2FA applications and two social media applications. Five 2FA applications were chosen based on popularity (downloads) and user ratings. This limited sample was intended to highlight security differences and common vulnerabilities rather than provide exhaustive coverage among commonly used applications.

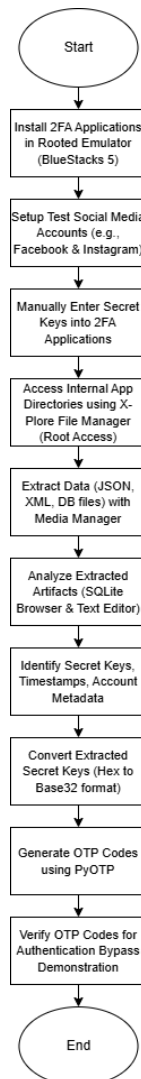
The data collection process is carried out using tools such as Media Manager and X-Plore File Manager to access the application directory. Media Manager is used to extract data directly from the virtual Android to the main computer, and X-Plore File Manager is used as a navigation tool to copy data from the application directory that requires root access. The data obtained is analyzed using SQLite Browser and Python (pyotp) to decrypt the OTP key and validate its ability to generate valid OTPs. The selection of this tool is based on considerations of compatibility with the Android-based testing environment and its ability to access, extract, and analyze data efficiently.

This research was entirely conducted in a controlled environment using test accounts, dummy data, and simulated scenarios within a rooted emulator (BlueStacks 5) to ensure user privacy and ethical standards. No real user data or devices were compromised during testing. The use of a rooted emulator was crucial for this study because it enabled unrestricted forensic-level access to internal data directories that typically remain protected, thus closely simulating potential attacker scenarios without harming real user devices.

To ensure the validity and repeatability of this research, all methodological stages have been transparently documented. The software specifications, tested application versions (as detailed in Table 1), and the tools utilized are explicitly described. By following the same documented procedure from environment configuration and data acquisition to artifact analysis other researchers should be able to replicate the findings and validate the identified vulnerabilities. This systematic approach ensures that the results are consistent and scientifically defensible.

Table 1. Tested 2FA Applications

Aplikasi 2FA	Versi	Unduhan	Rating
Google Authenticator	7.0	100M+	4.2
2FAS	5.4.9	1M+	4.2
Aegis Authenticator	3.3.4	500K+	4.5
Okta Verify	7.21.0	10M+	4.3
TOTP Authenticator	1.92	1M+	3.5

**Fig. 1.** Research Methodology Flowchart

This diagram illustrates the systematic workflow of the research, from the initial setup and data acquisition to artifact analysis and exploitation validation.

Scenario Creation and Testing

The first stage consists of testing 5 two-factor applications listed in Table 1, along with the number of downloads. The two social media applications selected for authentication are Facebook and Instagram. The 2FA applications used are selected based on their ratings and number of downloads. The reason is to compare the security systems between widely used applications, applications with high ratings, and applications that are less popular and have low ratings.

In order to conduct realistic testing and obtain all data completely and safely, this research was conducted using Bluestack 5 as a virtual Android that had been rooted first. All test applications used were downloaded from the Google Play Store. All 2FA app tests used both social media app services for authentication, namely Facebook and Instagram. It is important to note that the method used is by manually obtaining the key from the app, not by scanning a QR code that has been provided. Here are the steps and images of the Facebook app authenticated using the TOTP Authenticator app as shown in Figure 1 to demonstrate this process:

1. Set up all 2FA applications inside the virtual device.
2. Enable 2FA in social media apps (e.g., Facebook) and choose to use a pre-setup authentication app, such as TOTP Authenticator.
3. Copy the secret key generated by the social media app (e.g., Facebook) and enter it manually into the 2FA app to add the account. In some 2FA apps, you also need to add the account name or email along with the app service name.
4. After you have finished adding the account, the 2FA application will generate a six-digit code, which is needed to verify the social media application.
5. Log into the social media application using the six-digit code generated by the 2FA application.

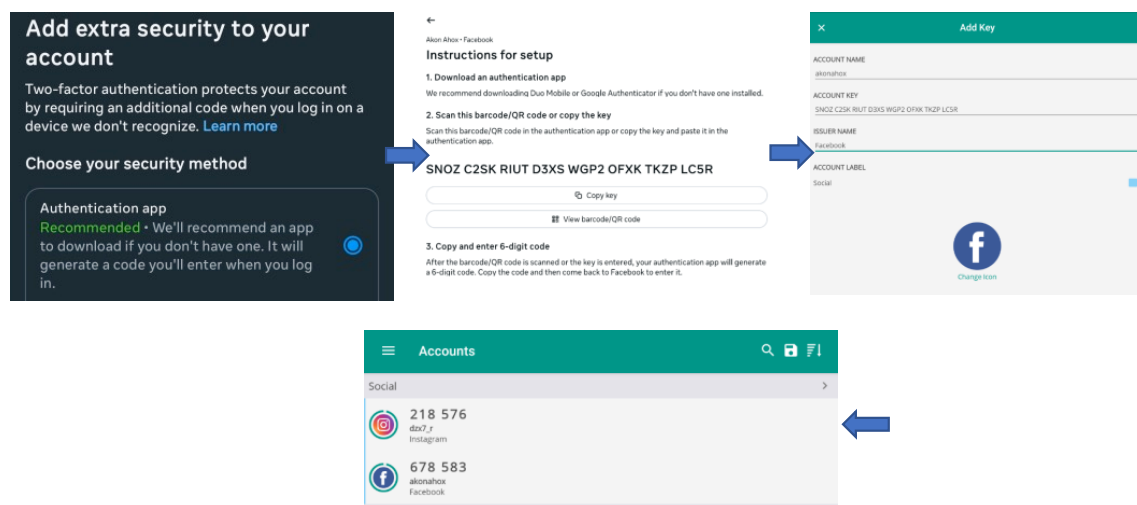


Fig. 2. Authentication Process

Data Acquisition

Data acquisition is an important step in the security analysis process to evaluate the storage and protection methods of data in Two-Factor Authentication (2FA) applications. This process is carried out carefully to ensure that the data taken is not changed or damaged. In this study, data was taken from five 2FA applications tested in Table 1, using a rooted Android virtual device (BlueStacks 5).

1. Acquisition Environment

In order for the acquisition process to run smoothly, this study uses a combination of specific software and hardware. BlueStacks 5 is used as an Android virtual device to provide an isolated environment and can be tested more flexibly. Root access is a primary requirement in this study in order to access the application directory and internal data in its

entirety. In addition, Media Manager is used to extract files from the virtual device to the main computer.

To facilitate navigation and management of files in the virtual device, the X-Plore File Manager application is used, which allows access to the system directory and application data. The extracted data is then stored on the main computer for further analysis.

2. Acquisition Procedure

The data acquisition process is carried out in three main steps. The first step is the installation and configuration of the application. In this stage, the 2FA application to be tested is downloaded and installed on a rooted Android virtual device. After that, two-factor authentication was enabled on the Facebook and Instagram accounts using the 2FA application, and the account and application information was saved for testing purposes. The second step is to access the application directory. X-Plore File Manager is used to open the system directory and find the data storage location of each 2FA application. The main directories identified include the databases folder containing files in database format, the shared_prefs folder containing files in XML format, and the files folder storing additional data used by the application in its operation. The last step in the acquisition process is data extraction using Media Manager. Data from the application directory on the virtual device is transferred to the main computer directly by selecting which files to move from the virtual device to the main computer, as shown in Figure 2. After all data from the application directory has been successfully extracted, the resulting files are then stored in the main computer for further analysis.

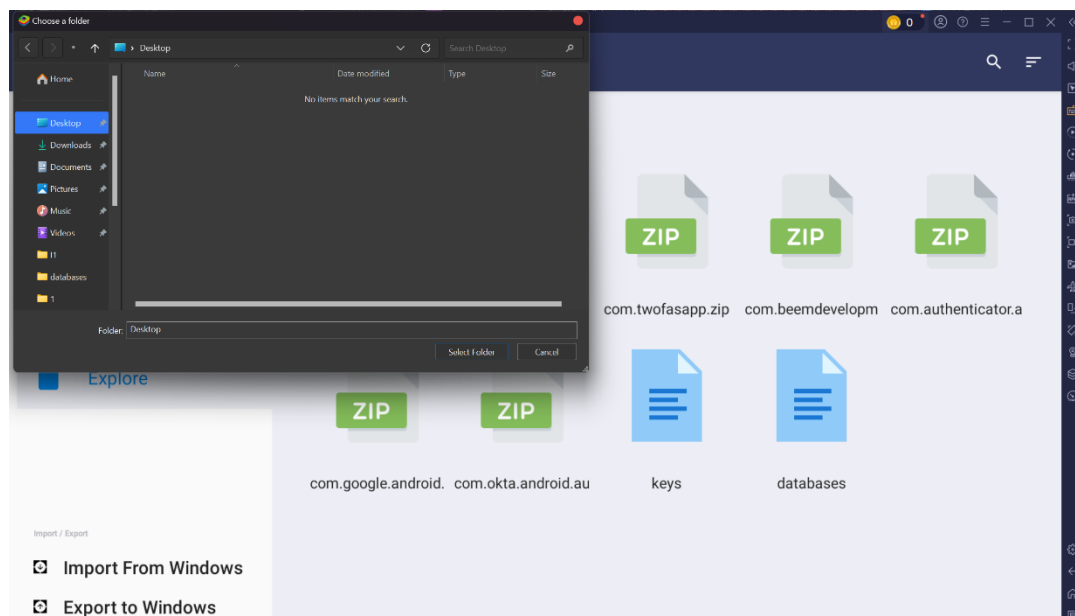


Fig. 3. Data Extraction Process

3. Data Collected

After the extraction process is complete, several types of files are found that usually contain important information from the 2FA application. One of them is the database, which contains tables with information such as secret keys, account names, timestamps, and other metadata. In addition, XML files contained in shared preferences also often store configuration data, including information about accounts connected to the application. Some applications also store data in the files folder, which can be JSON files or protocol buffers containing important artifacts.

In this data acquisition process, there are several challenges that must be faced. Some applications store secret keys in encrypted format, requiring additional approaches to

decrypt them. In addition, even though the virtual device has been rooted, some applications implement additional protection mechanisms that prevent access to their internal data. Some applications also use non standard data formats, such as protocol buffer (.pb) files, which require special tools to be read and analyzed.

Data Analysis and Eksploration

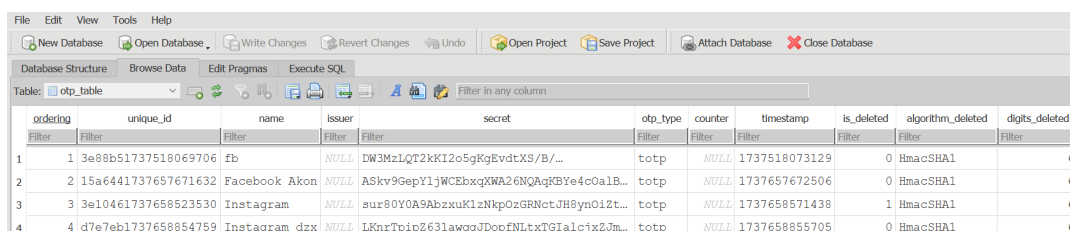
After the acquisition process is complete, the next step is to analyze the digital artifacts collected from the five two-factor authentication (2FA) applications. The main purpose of this analysis is to identify, decrypt, and validate the digital artifacts used in the One-Time Password (OTP) generation process, such as secret keys, timestamps, and other metadata. These exploration steps are carried out systematically to ensure that the artifacts found can be used in the bypass process or access recovery if needed.

In this process, various analysis tools are used to extract and read data stored in each application. SQLite Browser is used to read encrypted database files, while text editors are used to analyze configuration files stored in JSON or XML format.

Based on the exploration results, it was found that each application has a different data storage method, which affects the level of security and ease of accessing important information.

In Aegis Authenticator, data is stored in JSON format containing account information in plaintext. This JSON file contains various sensitive data, including 2FA type (TOTP or HOTP), UUID (Universal Unique Identifier), account name and issuer, secret key, encryption algorithm, OTP period (usually 30 or 60 seconds), and the number of OTP digits (usually 6 or 8 digits). With a neatly organized data structure in JSON format, this artifact is very easy to explore using a text editor without the need for additional tools.

Meanwhile, TOTP Authenticator stores data in XML format. This XML structure is more difficult to read directly than JSON because the data is not neatly arranged and organized. Information that can be extracted from the XML file includes timestamps (account creation time and last update), issuer and account name, secret key, algorithm used, and OTP period and length. However, because the data remains in plaintext, these artifacts can be easily read using a standard text editor. Google Authenticator has a different storage method than the previous two applications. Data in this application is stored in an encrypted database. This causes important artifacts such as secret keys and metadata to be inaccessible directly without first decrypting the database. To be able to access the database, a tool such as SQLite Browser or other additional description methods is required. After the database file was opened using SQLite Browser, artifacts such as unique ID, account name, secret key, OTP type, timestamp, and the algorithm used were found (see Figure 3). Although important artifacts were still found, compared to Aegis and TOTP Authenticator, Google Authenticator's storage method is more secure because the data cannot be accessed directly without first decrypting it.



ordering	unique_id	name	issuer	secret	otp_type	counter	timestamp	is_deleted	algorithm_deleted	digits_deleted
1	3e88b51737518069706	fb	NULL	DW3MzLT2kkI2o5gKgEvdXs/B/..	totp	NULL	1737518073129	0	HmacSHA1	6
2	15a6441737657671632	Facebook Akon	NULL	ASkv9GepY1jWCEbxqXWA26NqAqBYe4cOalB..	totp	NULL	1737657672506	0	HmacSHA1	6
3	3e10461737658523530	Instagram	NULL	sur80Y0A9AbzxuK1zNkpOzGRNctJH8ynO1Zt..	totp	NULL	1737658571438	1	HmacSHA1	6
4	d7e7eb1737658854759	Instagram dzx	NULL	LKnrTpip263lawqgJDopfNLTxTG1alcjxZJm..	totp	NULL	1737658855705	0	HmacSHA1	6

Fig. 4. Artifact from Google Authenticator

The 2FAS and Okta Verify applications also use encrypted database file formats. However, unlike Google Authenticator, when trying to open it using SQLite Browser, it was found that both applications implement additional encryption that requires a password to access it, as in Figure 4. This shows that the security system implemented by 2FAS and Okta Verify is tighter than other applications, so data exploration on these two applications requires a more in-depth method, such

as memory analysis or application communication interception, to find the right decryption method.

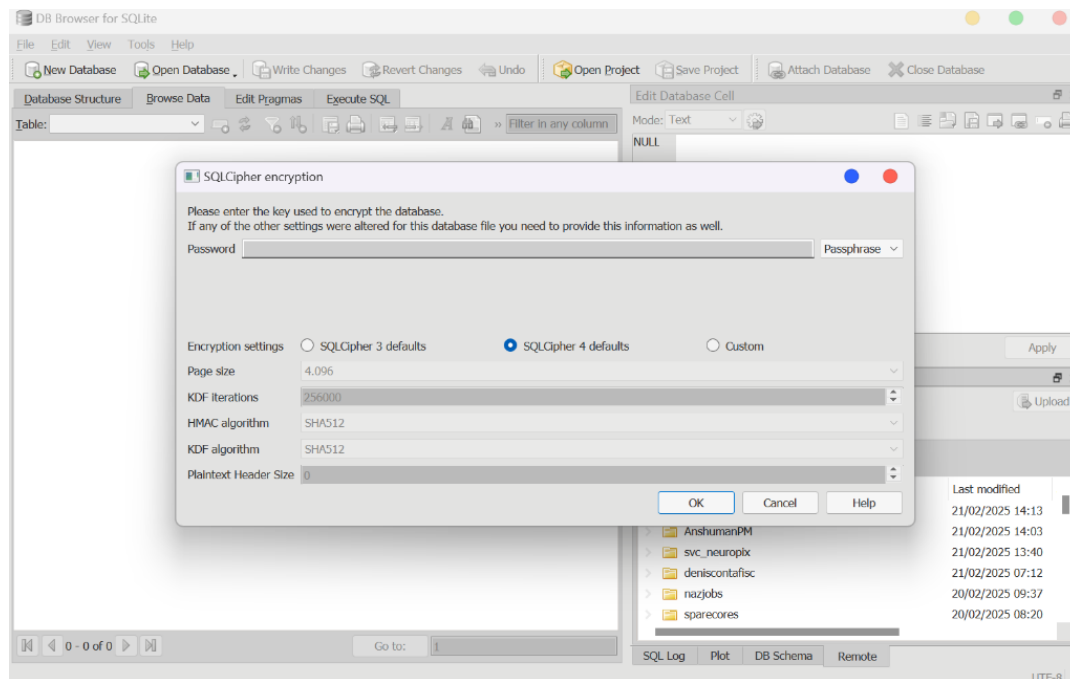


Fig. 5. Additional Encryption on the Okta Verify Database

RESULT AND DISCUSSIONS

Experimental Results

This study successfully identified data storage methods in the five 2FA applications tested in Table 1. The results of the extraction and security analysis show Out of the five tested applications, 2 out of 5 (40%) applications Aegis Authenticator and TOTP Authenticator stored secret keys in plaintext format, which significantly increases vulnerability to attacks. Meanwhile, the remaining 3 applications (Google Authenticator, 2FAS, and Okta Verify) utilized database encryption to store secret keys, but even these exhibited varying levels of protection. Specifically, 2FAS and Okta Verify incorporated additional password-based encryption, making them more resistant to direct extraction but still susceptible to advanced exploit methods.

This shows that applications using plaintext storage (Aegis Authenticator and TOTP Authenticator) have a significantly higher vulnerability to exploitation compared to those employing encryption methods (Google Authenticator, 2FAS, and Okta Verify), where encryption, although still vulnerable, presents a higher security threshold.

It can be seen in Table 2 that TOTP Authenticator and Aegis Authenticator have significant security weaknesses because they store secret keys in plaintext format, which makes them more vulnerable to exploitation. Meanwhile, Google Authenticator, 2FAS and Okta Verify use encryption on their database files and there is additional encryption in the form of a password on the 2FAS and Okta Verify applications (Figure 4) which complicates the data extraction process. While Google Authenticator uses encryption, the database is accessible with specific tools, demonstrating that encryption alone is insufficient without proper protection against extraction. On the other hand, 2FAS and Okta Verify implement additional layers such as password-based encryption, which makes them more resistant to direct extraction but still vulnerable to sophisticated exploitation if password security is compromised.

Table 2. Important Data Directory Paths and Artifact Storage Formats

2FA Application	Storage Format	Encryption Type	Directory Path
Google Authenticator	Database Encryption	Medium (Encryption)	data/data/com.google.android.apps.authenticator2/files/accounts/1/otp_database.db
2FAS	Database Encryption and Password	Strong (Password-Based)	data/data/com.twofasapp/databases/database-2fas
Aegis Authenticator	Plaintext JSON	None (No Encryption)	data/data/com.beemdevelopment.aegis/files/aegis.json
Okta Verify	Database Encryption and Password	Strong (Password-Based)	data/data/com.okta.android.auth/databases/com_okta_devices_storage_authenticator_datastore_sqlcipher_encrypted
TOTP Authenticator	Plaintext XML	None (No Encryption)	data/data/com.authenticator.authservice2/shared_prefs/TOTP_Authenticator_Preferences.xml

Bypass TOTP with PyOTP

In modern digital security systems, Time-based One-Time Password (TOTP)-based two-factor authentication (2FA) has become a widely used method to add a layer of protection to user account access. However, even though this system is designed to increase security, this study shows that the TOTP-based 2FA mechanism can still be exploited if the secret key used in the authentication process is not stored securely.

If an attacker manages to obtain the secret key from an application that does not implement strong protection or encryption, then the OTP code can be easily replicated without requiring direct access to the user's device. This is very dangerous because it allows attackers to bypass two-factor authentication and gain access to user accounts using only the login credentials they have obtained previously.

1. Security Concept in TOTP and Potential Exploitation

Before understanding how authentication bypass can be done, it is important to explain the basic concept of TOTP and how its security mechanism works. TOTP is a variant of Hash-based Message Authentication Code One-Time Password (HOTP) that uses the current time (UTC/GMT) as an input variable to generate a unique OTP code. This algorithm combines a secret key with a time value, which is then processed through the HMAC hashing function to generate an OTP code with a length of six to eight digits. This code is updated every certain period, usually 30 seconds, and is calculated almost simultaneously by the user's 2FA application and the service used, such as a social media account or other online platform [18].

When a user enables 2FA on a service, they are given a secret key wrapped in a QR code or plain text format that can be manually entered into the authentication app. The app then stores the key to generate an OTP code that can be used every time the user tries to log in

to their account. However, since all TOTP apps use the same HMAC algorithm, if an attacker manages to obtain the secret key from the app's file system, they can copy this key to another app or use a script to generate an identical OTP code.

In the research conducted, it was found that some authentication applications store secret keys in a format that is highly vulnerable to exploitation. Aegis Authenticator and TOTP Authenticator rely on outdated or simplistic encryption methods, which fail to meet modern security standards. These applications store secret keys in plaintext or with weak encryption, making it easier for attackers to exploit the system using basic analysis tools. In contrast, Google Authenticator, 2FAS, and Okta Verify implement stronger encryption, but the security can still be compromised if attackers have access to specific exploits or fail to use additional protection layers.

The experiment demonstrates the feasibility of bypassing TOTP security by extracting secret keys from applications that store them in plaintext or weakly encrypted formats. Once the secret key is obtained, the PyOTP library is used to replicate the OTP code, bypassing the 2FA mechanism. This method clearly highlights the potential for exploitation if secret keys are not properly protected during storage.

Based on an analysis of five tested authentication applications, it was found that Aegis Authenticator and TOTP Authenticator store secret keys in plaintext inside JSON or XML files. Both of these formats allow easy unauthorized access to the secret keys using only a text editor or basic analysis tools, thereby increasing the security risk for users [18].

Meanwhile, Google Authenticator, 2FAS, and Okta Verify store secret keys in the form of encrypted database files. However, after analyzing using SQLite Browser, it was found that the Google Authenticator database can be accessed with certain exploit methods. In contrast, the databases on 2FAS and Okta Verify have an additional layer of encryption that requires authentication before the file can be opened, making it difficult for direct extraction of the secret key[19]. This shows that despite efforts to protect the secret key, there are still loopholes that can be exploited by attackers, especially if the storage mechanism is not strong enough.

2. TOTP Bypass Implementation using PyOTP

To prove that the OTP code can be replicated using the extracted secret key, an authentication bypass experiment was conducted using the PyOTP library. This process begins by extracting the secret key from the application that stores it in plaintext or decryptable format. Once the secret key is successfully obtained, the next step is to convert it into a format that can be used by the TOTP algorithm.

The secret key obtained from the application is usually stored in hexadecimal format. Therefore, before being used in the TOTP algorithm, this key needs to be converted to base32 format first. This conversion is important because the TOTP algorithm requires a key in base32 format to be able to perform the hashing process correctly. After the key is converted, the PyOTP library is used to independently generate an OTP code, which can then be fed into the user account authentication system. The following code shows the process of converting and creating OTP for Facebook and Instagram accounts. using the OTP Library,

```
import pyotp
import base64

# Konversi Hex ke Base32
def hex_to_base32(hex_key):
    return
base64.b32encode(bytes.fromhex(hex_key)).decode('utf-8')
```

```
# Instagram
instagram_secret_hex =
"1CC9B8A90EAA197B9F613128EDA06773F40BD30C"
instagram_secret_base32 = hex_to_base32(instagram_secret_hex)
instagram_totp = pyotp.TOTP(instagram_secret_base32)
print("Instagram OTP:", instagram_totp.now())

# Facebook
facebook_secret_hex =
"C6DBB9B7D8BAA58F0D9DF8E50CF99A1B0997B689"
facebook_secret_base32 = hex_to_base32(facebook_secret_hex)
facebook_totp = pyotp.TOTP(facebook_secret_base32)
print("Facebook OTP:", facebook_totp.now())
```

The code leverages the PyOTP library to generate OTP codes based on the secret key that has been extracted from the authentication app. First, the key stored in hexadecimal format is converted to base32 using the `hex_to_base32()` function. Once converted, this key is used to generate OTP codes in real-time, which can then be fed into Facebook and Instagram's authentication systems.

3. Test Results and Security Impact

The test results show that the OTP code generated from the extracted key can be used to access accounts protected by two factor authentication. As long as the user's login credentials are also known, authentication bypass can be done easily without requiring direct access to the user's device.

These findings confirm that applications that store secret keys in plaintext are highly vulnerable to exploitation. The table below summarizes the effectiveness of the security mechanisms in place for each tested 2FA application in terms of secret key protection and the ease of bypassing the OTP mechanism,

Table 2. Important Data Directory Paths and Artifact Storage Formats

2FA Application	Key Storage	Encryption	Vulnerability to OTP Bypass
Google Authenticator	Database Encryption	Medium (Encryption)	Medium (Extraction possible with specific tools)
2FAS	Database Encryption and Password	Strong (Password-Based)	Low (Requires additional authentication password)
Aegis Authenticator	Plaintext JSON	None (No Encryption)	Medium (Extraction possible with specific tools)
Okta Verify	Database Encryption and Password	Strong (Password-Based)	Low (Requires additional authentication password)
TOTP Authenticator	Plaintext XML	None (No Encryption)	Very High (Easy access to secret key)

Furthermore, even though Google Authenticator, 2FAS, and Okta Verify use database encryption, the study shows that this protection can still be penetrated if the encryption can be broken or bypassed through memory analysis techniques.

The security of TOTP-based two-factor authentication depends heavily on how the secret key is stored by the application. Apps that do not implement strong encryption leave attackers open to extracting keys and replicating OTP codes. Therefore, 2FA app developers need to implement additional protections such as,

- 1) Using advanced encryption for the secret key storage database.
- 2) Utilize a hardware-based keystore system such as Android Keystore or iOS Keychain to stores the encryption key, making it more difficult to extract through reverse engineering.
- 3) Implement rate limiting on OTP-based login attempts to prevent brute-force attacks.
- 4) Adding additional protection mechanisms such as device binding, so that the OTP code can only be used on registered devices

Developers should immediately adopt stronger encryption methods, such as AES or RSA, to secure secret keys and account information. Additionally, integrating hardware-backed security mechanisms like Android Keystore or iOS Keychain is essential to prevent attackers from gaining access to sensitive data. This study proves that two-factor authentication is not a completely secure system if the secret key is not stored properly. Therefore, 2FA applications that store keys in an easily extracted format must immediately update their storage systems to avoid potential exploitation by irresponsible parties.

Comparison with Previous Research

In this study, the analysis was conducted on five two-factor authentication applications listed in Table 1. This study focuses on how these applications store secret keys and how possible exploits can be made to bypass authentication. To obtain more comprehensive results, this study is compared with a previous study conducted by Jessica Berrios et al. in the journal Factorizing 2FA: Forensic Analysis of Two-Factor Authentication Applications [17].

One of the main differences between the two studies is the number and scope of applications analyzed. The study by Berrios et al. analyzed fifteen 2FA applications running on various operating systems, including Android, iOS, and Windows. Meanwhile, this study only focused on five applications tested in an Android environment using BlueStacks 5. This difference indicates that the scope of this study is more limited in terms of platform variety but still relevant in revealing the weaknesses of secret key storage in popular applications used on Android. In addition, in terms of data acquisition methods, the previous study used a broader forensic approach, including disk, memory, and network traffic analysis. This study focuses more on security analysis and exploitation based on local file system analysis in a rooted Android emulator. Using tools such as Media Manager and X-Plore File Manager, this study was able to extract data from the internal directories of 2FA applications and identify how secret keys are stored in configuration files and application databases.

Another difference lies in the authentication bypass technique used. In the journal Berrios et al. [17], it was found that 14 out of 15 applications stored account information in plaintext or encrypted form, which allows exploitation through various reverse engineering methods. Meanwhile, this study shows that some applications still store keys in plaintext or with minimal encryption that allows the use of PyOTP to generate valid OTP codes. By using the extracted secret key, the OTP code can be replicated and used to bypass authentication without requiring the user's original device.

In terms of results and conclusions, Berrios et al.'s study emphasized that some applications have implemented advanced encryption to protect secret keys, but there are still weaknesses in the management of user information. This study strengthens these findings by showing that some

applications still use insecure storage methods, as well as providing evidence of exploitation through practical methods with PyOTP.

The reason for the weak encryption or plaintext storage in some applications, like Aegis Authenticator and TOTP Authenticator, may be due to the lack of proper key management strategies or reliance on outdated encryption methods that no longer meet modern security standards. In contrast, applications like 2FAS and Okta Verify, which use stronger encryption, demonstrate a more proactive approach to securing user data. However, even these applications are not immune to attacks if additional layers of protection, such as device binding or hardware-backed encryption, are not implemented. This indicates that despite improvements in the security of some 2FA applications, there are still gaps that can be exploited by irresponsible parties.

Based on this comparison, this study contributes with a more focused approach to the analysis of PyOTP-based exploits in a rooted Android environment. Unlike Berrios et al.'s broader forensic approach that primarily cataloged artifacts, our study places a stronger emphasis on demonstrating practical exploitation, such as bypassing the OTP mechanism using PyOTP. This shift from theoretical findings to real-world exploitation highlights the critical importance of application developers taking immediate action to secure secret keys and improve encryption practices. This study also confirms that weaknesses in secret key storage remain a critical issue in 2FA security. Therefore, additional measures are needed in the development of 2FA applications, such as implementing stronger encryption and protection mechanisms against unauthorized access to application system files.

CONCLUSIONS

Two-factor authentication (2FA) has become a widely used method to improve digital security, but this study shows that there are still significant weaknesses in the storage and management of secret keys in some 2FA applications. The analysis of the five tested applications revealed that not all applications apply the same security standards in protecting user authentication data.

Google Authenticator, 2FAS, and Okta Verify use database encryption to protect secret keys, but Google Authenticator still allows extraction of important artifacts with certain methods. Meanwhile, Aegis Authenticator and TOTP Authenticator store secret keys in plaintext format, which is very vulnerable to exploitation. Experiments have shown that OTP codes can be reconstructed using the PyOTP library if the secret key is successfully extracted, allowing authentication bypass without direct access to the user's device.

This study confirms that 2FA security depends not only on technology, but also on how applications manage secret keys. Applications that do not implement strong encryption open up opportunities for attackers to bypass authentication and access user accounts. Therefore, application developers must improve protection mechanisms, such as the use of hardware-based encryption like Android Keystore or iOS Keychain and the implementation of additional protection such as device binding and rate limiting to avoid brute-force attacks or spam on OTP-based login attempts that have the potential to threaten user cyber security [20]. Future research should expand analysis to other platforms (iOS, Windows), real-device forensic scenarios, and deeper memory analysis. These findings have implications for both organizational security policies and digital forensic practices, highlighting the importance of user awareness in selecting secure authentication applications, especially in 2FA apps. In addition, users are advised to choose 2FA applications that implement strong encryption and avoid applications that store data in plaintext. User awareness in choosing a secure application is very important to protect their accounts from cyber threats.

Overall, this study provides insight into the weaknesses of secret key storage in 2FA applications and emphasizes the need for improved security in two-factor authentication implementations. With improvements in data storage practices, 2FA can become a more reliable method of maintaining users' digital security. Ultimately, strengthening these practices can significantly enhance user trust and reduce the overall risk of authentication-related breaches.

REFERENCES

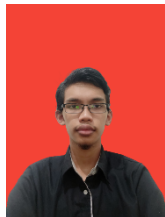
- [1] T. Mehra, "The Critical Role of Two-Factor Authentication (2FA) in Mitigating Ransomware and Securing Backup, Recovery, and Storage Systems," *Int. J. Sci. Res. Arch.*, vol. 14, no. 1, pp. 274–277, 2025. <https://doi.org/10.30574/ijrsra.2025.14.1.0019>
- [2] F. P. E. Putra, A. Zulfikri, G. Arifin, R. M. Ilhamsyah, and others, "Analysis of phishing attack trends, impacts and prevention methods: literature study," *Brill. Res. Artif. Intell.*, vol. 4, no. 1, pp. 413–421, 2024. <https://doi.org/10.47709/brilliance.v4i1.4357>
- [3] D. Patel, D. Trivedi, U. Raval, and A. Dennisan, "2F-Authsys: A hyperlocal two-factor authentication system using Near Sound Data Transfer," *J. Appl. Res. Technol.*, vol. 22, no. 2, pp. 197–205, 2024. <https://doi.org/10.22201/icat.24486736e.2024.22.2.2244>
- [4] M. R. Suresh, K. Balachander, S. L. Varman, and K. S. S. Faris, "Two Factor Authentication by Using Decentralized File Storage System," 2024. <https://doi.org/10.21203/rs.3.rs-4316141/v1>
- [5] T. Taffese, F. Chavez, A. Fernandez-Reyes, and M. Byrne, "Redesigning and evaluating interfaces for two-factor authentication: performance of younger and older adults," 2025, [doi: 10.31234/osf.io/vtx7u_v1](https://doi.org/10.31234/osf.io/vtx7u_v1).
- [6] M. Az, S. F. Pane, and R. M. Awangga, "Cryptography: Perancangan Middleware Web Service Encryptor menggunakan Triple Key MD5, Base64, dan AES," *J. Tekno Insentif*, vol. 15, no. 2, pp. 65–75, 2021. <https://doi.org/10.36787/jti.v15i1.497>
- [7] K. Liu et al., "A robust and effective two-factor authentication (2FA) protocol based on ECC for mobile computing," *Appl. Sci.*, vol. 13, no. 7, p. 4425, 2023. <https://doi.org/10.3390/app13074425>
- [8] M. D. Tomić and O. M. Radojević, "Implementation of two-factor user authentication in computer systems," *Vojnoteh. Glas. Tech. Cour.*, vol. 72, no. 1, pp. 170–191, 2024. <https://doi.org/10.5937/vojtehg72-48081>
- [9] C. Cornelius and T. Simon, "Investigate and Evaluate the Security Measures Commonly Used in Electronic Banking Transactions in Zambia and Possible Solutions," *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 5, pp. 9077–9082, 2023. <https://www.doi.org/10.56726/IRJMETS41115>
- [10] B. D. Kurniawan, M. A. Rosid, I. A. Kautsar, and N. E. Pratama, "Rancang Bangun Library Web Token untuk Enkripsi HTTP Data Menggunakan Eksklusif-OR (XOR)," *Phys. Sci. Life Sci. Eng.*, vol. 1, no. 1, p. 14, 2023. <https://doi.org/10.47134/pslse.v1i1.164>
- [11] B. Anwar, R. Kustini, and I. Zulkarnain, "Penerapan Algoritma RSA (Rivest Shamir Adelman) Untuk Mengamankan Nilai Siswa SMP HKBP P. Bulan," *J. Teknol. Sist. Inf. dan Sist. Komput. TGD*, vol. 4, no. 1, pp. 88–91, 2021. <https://doi.org/10.53513/jsk.v4i1.2623>
- [12] Y. Sun, S. Zhu, Y. Zhao, and P. Sun, "Let Your Camera See for You: A Novel Two-Factor Authentication Method against Real-Time Phishing Attacks," *arXiv Prepr. arXiv2109.00132*, 2021. <https://doi.org/10.48550/arXiv.2109.00132>
- [13] J. Haddad, N. Pitropakis, C. Chrysoulas, M. Lemoudden, and W. J. Buchanan, "Attacking Windows Hello for Business: Is It What We Were Promised?," *Cryptography*, vol. 7, no. 1, p. 9, 2023. <https://doi.org/10.3390/cryptography7010009>
- [14] A. Aldahmani, B. Ouni, T. Lestable, and M. Debbah, "Cyber-Security of Embedded IoTs in Smart Homes: Challenges, Requirements, Countermeasures, and Trends. *IEEE Open Journal of Vehicular Technology*, 4, 281-292." 2023. <https://doi.org/10.1109/OJVT.2023.3234069>
- [15] S. J. Murdoch and A. Abadi, "A Forward-secure Efficient Two-factor Authentication Protocol," *arXiv Prepr. arXiv2208.02877*, 2022. <https://doi.org/10.48550/arXiv.2208.02877>
- [16] E. Ahmadih and N. El Madhoun, "Comparative E-Voting Security Evaluation: Multi-Modal Authentication Approaches," in *2024 6th International Conference on Blockchain Computing and Applications (BCCA)*, 2024, pp. 250–254. <https://doi.org/10.1109/BCCA62388.2024.10844450>
- [17] A. Sofian et al., "Enhancing authentication security: analyzing time-based one-time

- password systems,” Int. J. Comput. Technol. Sci., vol. 1, no. 3, pp. 56–70, 2024, doi: 10.62951/ijcts.v1i3.25. <https://doi.org/10.62951/ijcts.v1i3.25>
- [18] R. O. Okeke and S. O. Orimadike, “Enhanced Cloud Computing Security Using Application-Based Multi-Factor Authentication (MFA) for Communication Systems,” Eur. J. Electr. Eng. Comput. Sci., vol. 8, no. 2, pp. 1–8, 2024. <https://doi.org/10.24018/ejece.2024.8.2.593>
- [19] J. Berrios, E. Mosher, S. Benzo, C. Grajeda, and I. Baggili, “Factorizing 2FA: Forensic analysis of two-factor authentication applications,” Forensic Sci. Int. Digit. Investig., vol. 45, p. 301569, 2023, doi: 10.1016/j.fsidi.2023.301569.
- [20] S. Pane and D. J. W. Ikram, “Deteksi Spam Bot Pada Komentar Youtube: Tinjauan Literatur Sistematis,” Comput. Sci. Res. Its Dev. J., vol. 15, no. 2, pp. 103–123, 2023. <https://www.doi.org/10.22303/csrid.15.2.2022.103-123>

AUTHORS BIBLIOGRAPHY



SYAFRIAL FACHRI PANE was born in Medan, North Sumatra in April 1988. He obtained his bachelor of informatics degree from Pasundan University and master of informatics from Bina Nusantara University, Bandung, in 2019 and 2021, respectively. Currently, he is pursuing his doctoral program at Telkom University, Bandung. He is involved in data science and machine learning research. He is also a lecturer at the University of Logistics and International Business (ULBI), Bandung. His research interests include data analytics and machine learning. His research dissertation focuses on Hybrid Multi-objective Metaheuristic Machine Learning for Pandemic Modelling.



DZIKRI IZZATUL HAQ was born in Bandung, West Java, in March 2004. He is currently an undergraduate student in the Informatics Engineering program at the University of Logistics and International Business (ULBI). As part of his Applied Bachelor's (D4) studies, he is developing a strong interest and specialization in the field of backend development. His academic journey began after graduating from the reputable SMAN 9 Bandung, which prepared him for his higher education pursuits in technology.



M AMRAN HAKIM SIREGAR was born in Medan, North Sumatra, in July 1997. He obtained his bachelor's degree in informatics engineering from Politeknik Pos Indonesia and master's degree in computer science from STMIK LIKMI, Bandung, in 2019 and 2022, respectively. Currently, he is a lecturer at Universitas Mitra Bangsa, Jakarta. He has several publications of scientific papers on data science implemented using several programming languages.