Analysis of WhatsApp data-at-rest security and encryption usage*

Leila Rzayeva[1,*,†], Abilkair Imanberdi[1,†], and Yerassyl Yermekov[2,†]

[1] Research and Innovation Center "CyberTech," Astana IT University, 55/11 Mangilik El ave., Business center EXPO C1, Astana, Kazakhstan
[2] L. N. Gumilyov Eurasian National University, 2 Satbaev str., ENU Educational bl., Astana, Kazakhstan

Abstract

This paper evaluates the data-at-rest security of the current WhatsApp version for Android. Personal data protection in transit and at rest is important for ensuring a comprehensive security system, especially on mobile devices, which are one of the most common digital forensics targets. The evaluation was conducted using an Android phone running Android 6 and the latest version of WhatsApp, and a computer with the Linux Mint operating system. The results showed a lack of database encryption in the app-specific internal storage and the usage of insecure key storage. Chat backups were encrypted, although the key is not stored in the secure storage. We conclude that the level of local data protection of the WhatsApp Android client is insufficient for forensic analysis prevention, although the ordinary user has no access to the encryption keys without obtaining root permissions.

Keywords

data-at-rest encryption, personal data, instant messaging, WhatsApp, database encryption, Android, forensics.

1. Introduction

The vast majority of instant messaging applications utilize local data storage for the long-term retention of user information, including media content, text messages, and configuration files. Some instant messaging applications, such as Telegram, minimize the use of long-term local data storage. In order to achieve a sufficient security level, long-term local data storage should be secured to counteract unauthorized access and forensic analysis attempts.

The most efficient method of ensuring the proper security of local data storage is by the use of cryptographic algorithms on local databases and media vaults. Furthermore, mobile operating systems, such as Android and iOS, also provide secure data storage for applications, which is not accessible to users directly or through other applications. Each application can only use its data sub-catalog and is unable to traverse to other data sub-catalogs.

The developers of WhatsApp, Meta, claim to provide "Secure and Reliable Free Private Messaging and Calling" with this application. Despite that, numerous lawsuits were filed over the last few years to question the security of the platform. A recent class-action complaint, filed in the San Francisco division of the Northern District of California [1], claims that Meta Platforms Inc and WhatsApp LLC can access end-to-end encrypted messages using internal tools, which violates the Wiretap Act (18 U.S.C., section 2510 et seq.) and multiple California state acts. Another lawsuit [2] was filed in the Northern District of California by Attaullah Baig, the former head of security at WhatsApp, claiming that Meta engineers had excessive access permissions and that the company is in breach of the FTC's 2020 privacy order. Both lawsuits are in early stages as of February 2026.

Previous research concentrated more on the data-in-transit and largely omitted the data-at-rest security of instant messaging applications [3]. Furthermore, the previous research in the field of WhatsApp backup encryption failed to describe the principles of database encryption.

* CSDP'2026: Cyber Security and Data Protection, May 7, 2026, Lviv, Ukraine
* Corresponding author.
† These authors contributed equally.
l.rzayeva@astanait.edu.kz (L. Rzayeva); a.imanberdiyev@astanait.edu.kz (A. Imanberdi); 990726350870@enu.kz (Y. Yermekov)
0000-0002-3382-4685 (L. Rzayeva); 0009-0005-6144-2392 (A. Imanberdi); 0009-0001-3957-4787 (Y. Yermekov)

The aim of the research is to systemize the knowledge about the encryption algorithms used in the WhatsApp Messenger and test the security of WhatsApp's local data and backup encryption on the Android platform.

2. Literature review

The problem of forensic analysis of instant messaging applications has been actively researched over the last 10 years, although most research is focused on data protection in transit rather than at rest. The research is particularly active in the field of analysis of mobile clients' data.

Shortall and Azhar (2015) [4] used UFED and Oxygen Detective software packages to extract WhatsApp data from three specimens of mobile devices with Windows Phone 8.1, Android 5.0.1, and iOS 8.3 operating systems. The results showed that the extraction of data from Android and iOS devices was successful, unlike Windows Phone 8.1. The paper provides practical techniques for forensic recovery of Windows Phone 8.1 WhatsApp data.

Kaushik and Katara (2022) [5] utilized Oxygen Detective and a custom Python application in order to extract and analyze WhatsApp backup data. However, the quality of the research is severely impaired due to the use of the old WhatsApp client version with a known vulnerability, which allows rootless data extraction using the Android Debug Bridge toolkit. This vulnerability was patched in subsequent versions.

Iqbal and Riadi (2019) [6] applied the NIST SP 800-86 four-stage plan for the forensic examination of WhatsApp data. A rooted device was used to extract the crypt12 database and its key.

Utomo et al. (2023) [7] also applied the NIST model for the forensic evaluation of WhatsApp data for the desktop web version of the messenger application. A conversation between two users was simulated; the data was acquired using the FTK imager utility and analyzed with The Sleuth Kit and Autopsy. The encryption version was detected as Crypt8, although newer encryption versions may be used in future versions of the web application.

Yadav et al. (2020) [8] utilized Belkasoft Evidence Center for the acquisition of a crypt12 backup sample and the encryption key. This study fails to compare the used software solution to its more mature competitors, such as UFED, Oxygen Detective, et cetera.

Jha (2025) [9] presented an investigation framework for deleted message recovery. However, this research falls short in presenting results of implementing said framework in existing forensic systems and has no practical meaning.

In contrast to previous research, Davies et al. (2023) [10] provided a fundamental security analysis of the new end-to-end encrypted backup protocol (which was named WBP — WhatsApp Backup Protocol), based on an OPAQUE asymmetric key exchange protocol. The WBP in its current implementation has no obvious security flaws; however, a compromised server may reduce the security of the key exchange algorithm by increasing the number of possible password guesses.

3. Android application data storage architecture

Android applications typically store their data in one or more locations on the device. The storage is separated between the internal and external memory in terms of the physical architecture. The internal storage, contrary to general understanding, is not accessible by the user and does not contain user-accessible data. The internal storage represents the "/data" partition of the eMMC memory. The external storage (also called "General Storage" in newer Android versions) consists of the "/storage" partition of the eMMC memory and any external storage devices, such as SD cards, USB mass storage devices, et cetera. The external storage is generally visible and accessible to the user.

In terms of the logical architecture, the data storage is split into the app-specific storage, shared storage, preferences registry, and a separate database storage.

3.1. App-specific storage

The app-specific storage is used for files that should not be accessed by other applications and/or device users. This storage section is located on the separate /data partition of the internal or external device memory, and is not accessible by traversing the shared storage location. Each application has access only to the catalog with its package name (e.g., WhatsApp's package name is com.whatsapp). The access to the app-specific storage is abstracted using the Android API. The app-specific storage is not persistent and is cleared after the application is uninstalled.

The internal app-specific storage is encrypted using the full-disk or file-based encryption methods, depending on the device and its software version, which increases its security against physical data extraction attacks.

Larger files (such as cache files, streaming assets, etc) should be saved in the external app-specific storage, typically located in the "Android" subcatalog of the shared storage. The external app-specific storage catalog is visible to the user, but its structure is not accessible for listing, reading, and writing. Earlier Android versions did not restrict access to this storage section. The external app-specific storage is encrypted together with the user data, as they are located on the same memory partition.

3.2. Shared storage

The shared storage is located in the external data partition and is used for the storage of data, which may or should be used by other installed applications or users. This storage section is accessible by any application, user, and the MTP daemon.

A new scoped access restriction model for applications was introduced in Android 12. Applications have limited storage access by default, typically restraining access to the data of other applications and media files. The access can be broadened by the user's explicit permission to allow access to all files.

3.3. Preferences registry

The global preferences registry is a key-value storage for applications to save custom data (typically, user preferences) in an organized manner. The first implementation of the registry, SharedPreferences, was introduced in Android API level 1, which corresponds to the Android 1.0 version. The SharedPreferences API provides methods for creating, editing, and deleting entries. SharedPreferences uses XML files for settings storage, which are stored in the shared_prefs subcatalog of the internal app-specific storage.

In late 2020, a replacement Jetpack DataStore API was introduced, offering asynchronous access, transactions, thread safety, type safety, improved error and corruption handling, and migration assistance.

As of 2026, neither of the frameworks offers data encryption and relies on data encryption on the application side.

3.4. Database storage

The separate database storage was introduced in May 2018 with the Jetpack Room API, which provides an abstraction layer over SQLite and offers unified database management, in contrast to previous approaches with manual database handling and querying. The Jetpack Room API does not provide any encryption by itself and relies on external encryption methods, such as SQLCipher.

4. File-based database encryption methods

The most common file-based (also called "serverless") relational database management system is SQLite, which offers high levels of integration and compact implementation of a full-featured relational database management system. Currently, there are two production-ready encryption

methods for SQLite: a free and open-source SQLCipher and a closed-source SEE (SQLite Encryption Extension). Both provide high levels of security against brute-force attacks by using the AES encryption and PBKDF2 key derivation algorithms, although a raw key can be used to perform cryptographic operations.

SQLCipher is more commonly used in various applications, where the database is stored locally on both secure and non-secure storage locations. The SEE is significantly less common due to its paid and closed-source nature.

Developers may use custom encryption procedures for SQLite database files, as they have no substantial difference compared to other files.

5. WhatsApp backup storage encryption

5.1. Early encryption attempts (crypt5, crypt8)

One of the earliest known attempts to implement backup encryption was in 2014, when crypt5 database encryption was introduced. This encryption scheme used a hybrid key derivation approach based on a static base key XOR-ed with a dynamic component derived from the Google account username using an MD5 message digest [11]. The initialization vector is static across all instances. Databases were encrypted using the AES-192 algorithm in CBC mode with the PKCS#5 padding scheme. This key derivation scheme has obvious flaws, which allowed for the easy decryption of data backups with prior knowledge of the username only (Figure 1).

The crypt8 encryption scheme was introduced in 2015. The main difference of this scheme is an improved key management and derivation scheme, which was not reverse-engineered. The encryption algorithm was changed from AES-192-CBC to AES-256-CBC [12] (Figure 2). The encryption key and IV are stored in a single file, located in the internal device memory and easily accessible with root permissions.

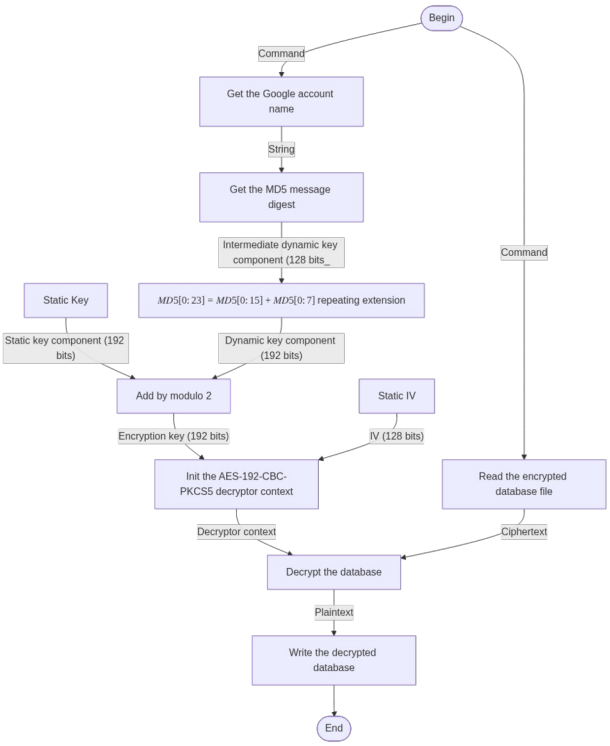

Figure 1: The diagram of the crypt5 algorithm decryption process.

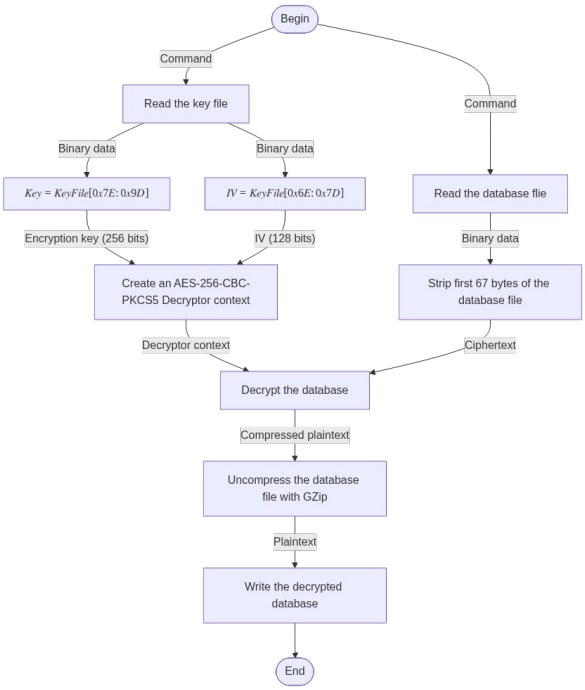

Figure 2: The diagram of the crypt8 algorithm decryption process.

5.2. Modern server-tethered encryption (crypt12, crypt14)

More modern backup encryption schemas include crypt12 and crypt14. These schemas utilize the AES-256-GCM encryption algorithm [11] in order to increase performance compared to the legacy CBC mode, which does not support parallelization. Furthermore, the usage of the GCM mode adds data authenticity verification capabilities due to its AEAD nature [15].

Crypt12 and crypt14 differ in the database header structure, which has no significant impact on the encryption security [13]. For example, the crypt12 database uses a raw header format with fixed offsets (Figure 3), while the crypt14 database uses a Protobuf structure (Figure 4). Key structures are identical across crypt12 and crypt14 and are stored in the internal memory as Java serialized objects (Figure 5).

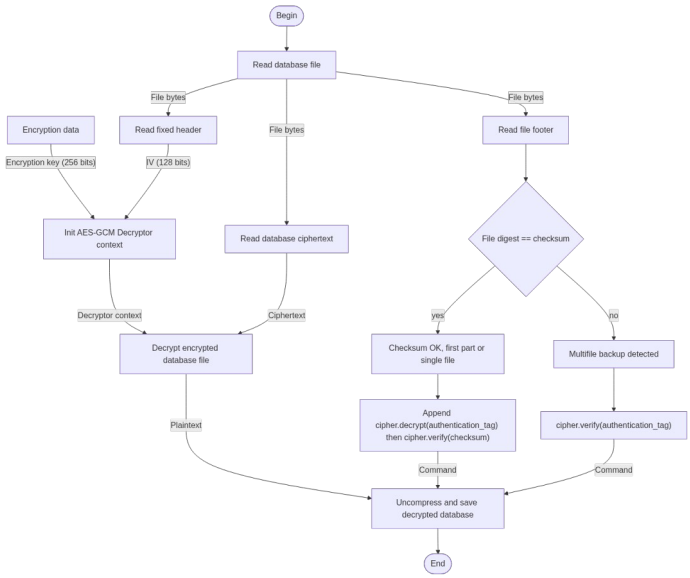

Figure 3: The diagram of the crypt12 algorithm decryption process.

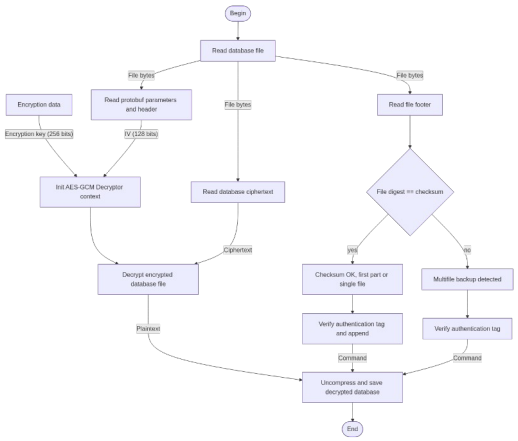

Figure 4: The diagram of the crypt14 algorithm decryption process.

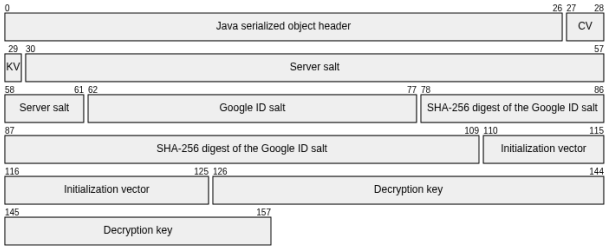

Figure 5: The structure of the common crypt12 and crypt14 key file.

5.3. End-to-end encrypted backups (crypt15)

A new end-to-end backup encryption mechanism was introduced in 2023. Unlike the previous backup system, which stored the encryption key on the server, the new system prompts the user to manually create and store the encryption key on-premises [13]. Thus, neither the server nor the cloud provider has the knowledge of the key, which inherently increases the security level. The key is stored using the hardware security module in case of a user password used for backup encryption, or in internal memory in case of an auto-generated key.

The encryption specifications are similar to the crypt14 format (Figure 6). The encryption key consists of 32 bytes and has no additional data.

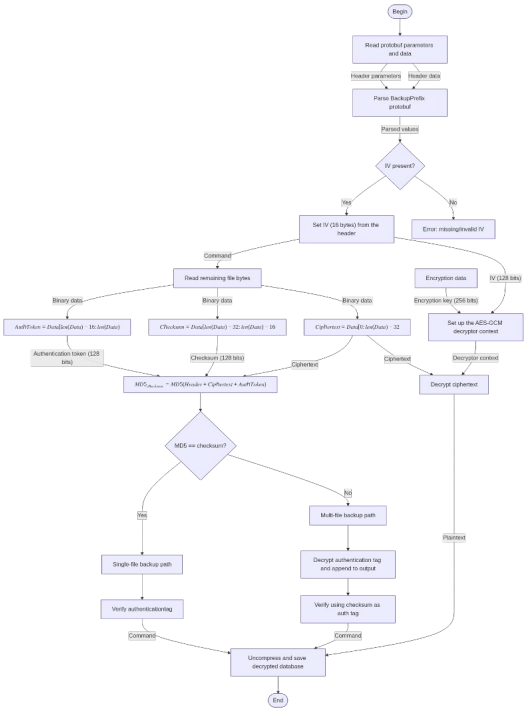

Figure 6: The diagram of the crypt14 algorithm decryption process.

6. Experimental evaluation of the WhatsApp data-at-rest security

6.1. Evaluation methodology

The evaluation of data-at-rest security of WhatsApp will consist of three steps:

- Device initialization;
- Data retrieval;
- Data analysis.

In order to complete the evaluation, the following list of prerequisites is needed:

- An Android smartphone with installed root access and a WhatsApp client;
- A Linux personal computer with Android Debug Bridge toolkit;
- Supplementary software (hex editors, database viewers, decryption utilities, et cetera).

The following hardware and software products were used to complete the experimental evaluation:

- A Huawei P9 Lite smartphone (VNS-L21, firmware VNS-L21C432B170 , Android 6.0, EMUI 4.1.2);
- A ThinkPad T14 Gen 2 with Linux Mint distribution and ADB version 1.0.41;
- ImHex hexadecimal editor [16], DB Browser for SQLite [17], and wa-crypt-tools [13] software packages.

6.2. Evaluation process and results

6.2.1. Device initialization

The device was rooted using the Magisk toolkit [18] After the rooting process, the latest version of WhatsApp was installed on the device and activated with a secondary phone number. A denylist entry for WhatsApp was added to prevent possible root detection by the application. A test conversation was added to the device under evaluation (Figure 7).

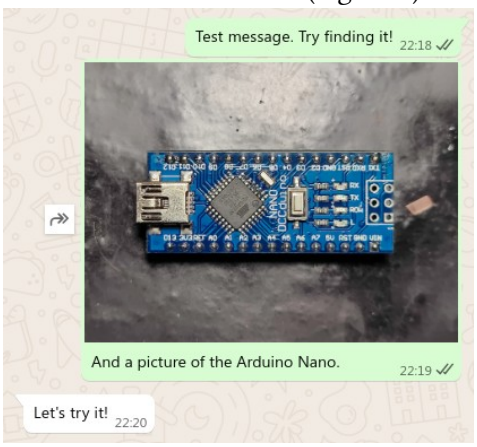

Figure 7: The test conversation sample.

Next, the "USB Debugging" was enabled in Developers settings menu for data extraction through the ADB interface.

6.2.2. Data retrieval

A local data image of the "/data/data/com.whatsapp" internal app-specific storage catalog was created using the tar application with root access. The image archive was transferred to the Linux PC using the "adb pull" command and unarchived using the "tar" command.

Local copies of server-encrypted backups are stored in the shared storage at "/storage/emulated/0/WhatsApp". An identical extraction process was applied to local backups, both server-encrypted and end-to-end-encrypted, as they are stored in the same location.

6.2.3. App-specific data storage analysis

During the directory tree inspection (Figure 8), the following databases were found:

- account_switcher.db (user accounts list in case of adding multiple accounts in a client);
- axolotl.db (E2EE key storage);
- chatsettings.db (individual and global chat settings);
- commerce.db (user purchases data);
- companion_devices.db (other devices, connected to the account);
- daily_metrics.db (user activity metrics);
- _jobqueue-WhatsAppJobManager (a temporary registry of asynchronous worker jobs);
- location.db (device location tracking history);
- media.db (media file IDs storage);
- msgstore.db (text message history);
- payments.db (WhatsApp Payments money transfer history);
- stickers.db (installed and recently used stickers);
- sync.db (message synchronization log);
- wa.db (a list of contacts, chats, groups, bot profiles, etc).

Figure 8: App-specific storage tree structure (directories-only view).

File analysis showed that the databases are presented in an unencrypted form, complete with a shared memory file and a write-ahead log, which allows for reproducing the history of text conversations. Text messages are saved in the "messages" table of the "msgstore.db" database file (Figure 9).

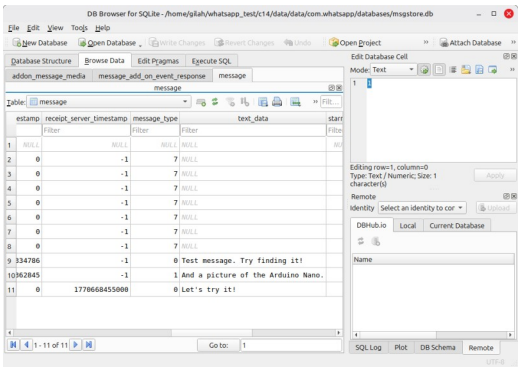

Figure 9: Message table contents of the app-specific copy of the message storage database.

Backup encryption keys are located at “com.whatsapp/files/key” for server-encrypted backups and “com.whatsapp/files/encrypted_backup.key” for end-to-end-encrypted backups.

Other potentially useful data include server session cache (com.whatsapp/cache/SSLSessionCache), profile pictures of contacts (com.whatsapp/files/Avatars), log files (com.whatsapp/files/Logs), and shared preferences (com.whatsapp/shared_prefs).

6.2.4. Server-encrypted backup analysis

Current server-encrypted backups utilize the crypt14 algorithm, as indicated by the backup file names (Figure 10).

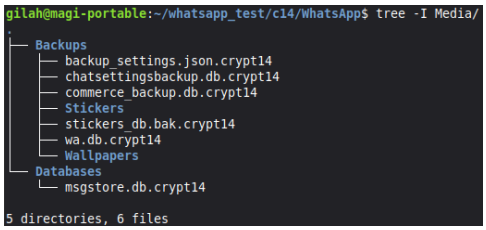

Figure 10: The structure of the server-encrypted backup (media files excluded).

A wadecrypt utility was used to perform file decryption. The resulting backup files contain data schemas, identical to files, found in the internal app-specific storage, except for the "backup_settings.json" file, which is specific to backups and contains backup settings (Figure 11).

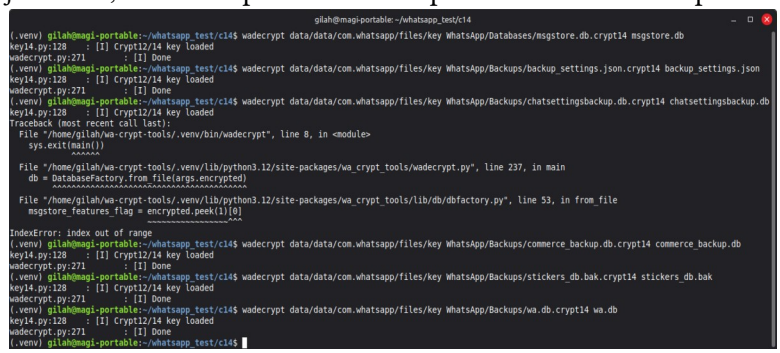

Figure 11: Server-encrypted backup decryption process.

One decryption error happened during the dectyprion of the “chatsettingsbackup.db” file. The "msgstore.db" backup file contains text messages identical to the "msgstore.db" app-specific storage file (Figure 12). Media files are stored without encryption (Figure 13).

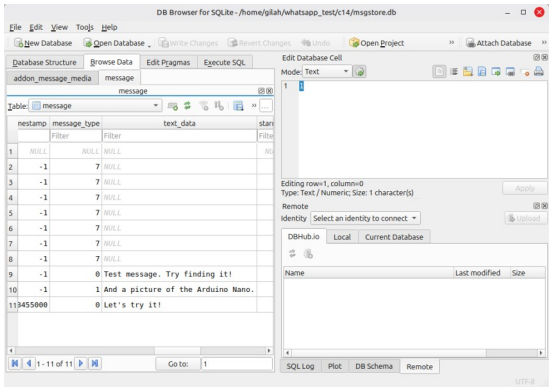

Figure 12: Message table contents of the server-encrypted backup copy of the message storage database.

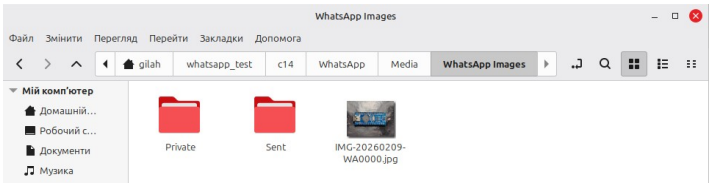

Figure 13: Server-encrypted backup media storage.

6.2.5. End-to-end-encrypted backup analysis

End-to-end-encrypted backups utilize the crypt15 algorithm (Figure 14), which was described in Section 5.3.

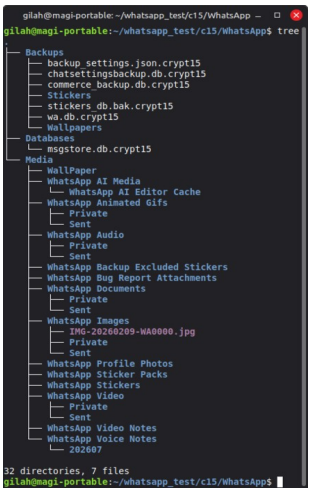

Figure 14: The structure of the end-to-end-encrypted backup.

Databases were successfully decrypted using the "encrypted_backup.key" file (Figure 15) and are identical to previous files (Figure 16).

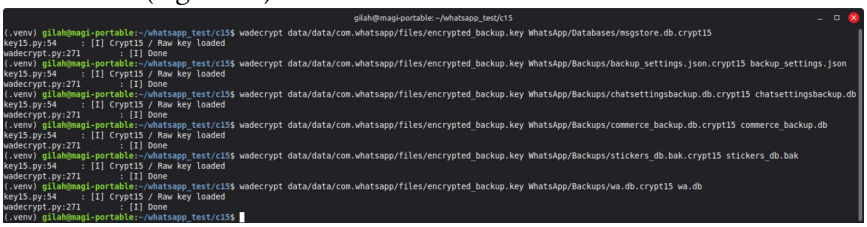

Figure 15: End-to-end-encrypted backup decryption process.

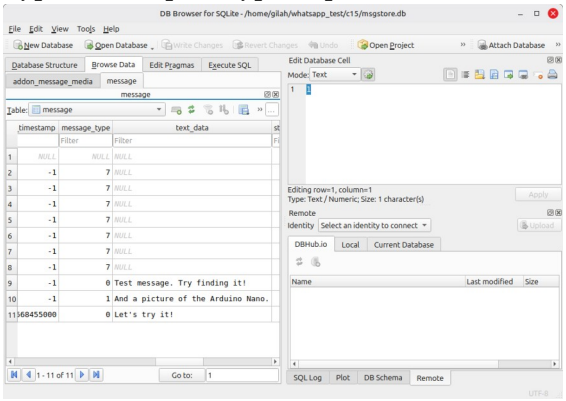

Figure 17: Message table contents of the end-to-end-encrypted backup copy of the message storage database.

Conclusions

As instant messaging applications become a staple of modern communication, it is essential to evaluate the data security both in transit and at rest. While the in-transit security of WhatsApp has been widely researched and discussed, its at-rest security is often underestimated. In this research, a comprehensive analysis of WhatsApp's encryption protocols was conducted.

The analysis of backup encryption algorithms shows no significant vulnerabilities in current versions of algorithms, although previous versions used weak key derivation schemes with static keys and initialization vectors. The analysis of app-specific internal storage showed no additional security implemented besides the system-provided protection mechanisms, such as access control and partition encryption. Server-encrypted and end-to-end-encrypted backups can be easily decrypted with obtaining the locally-stored encryption key, which can be accessed by using root access or other methods.

The lack of secondary local database encryption and plaintext encryption key storage in local memory instead of the hardware-based Android Keystore API poses a serious threat to the security of WhatsApp client instances on Android operating systems. Notwithstanding the research results, the normal user access is limited by standard Android security features, such as different mount locations and system-level data encryption.

Future research into WhatsApp security may focus on rootless key extraction tactics for backup decryption or other methods of app-specific storage extraction.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

[1] Dawson et al. v. Meta Platforms, Inc., N.D. Cal., 3:26-cv-00751-RFL (USA).

[2] Baig v. Meta Platforms, Inc. et al, N.D. Cal., 4:25-cv-07604-YGR (USA).

[3] O. Harasymchuk, et al. (2025) Modern Methods of Ensuring Information Protection in Cybersecurity Systems using Artificial Intelligence and Blockchain Technology. Kharkiv: Technology Center PC, p. 132 doi:10.15587/978-617-8360-12-2.

[4] A. Shortall, M. Bin Azhar, Forensic Acquisitions of WhatsApp Data on Popular Mobile Platforms, in: 2015 Sixth International Conference on Emerging Security Technologies (EST), IEEE, 2015. doi:10.1109/est.2015.16.

[5] K. Kaushik, Y. Katara, Forensic Analysis of WhatsApp Chat Data, in: 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), IEEE, 2022. doi:10.1109/icrito56286.2022.9965028.

[6] M. Iqbal, I. Riadi, Forensic WhatsApp based Android using National Institute of Standard Technology (NIST) Method, Int. J. Comput. Appl. 177.8 (2019) 1-7. doi:10.5120/ijca2019919443.

[7] D. Utomo, Y. Prayudi, E. Ramadhani, Forensic Web Analysis on The Latest Version of Whatsapp Browser, J. Comput. Netw. Archit. High Perform. Comput. 5.1 (2023) 359-367. doi:10.47709/cnahpc.v5i1.2286.

[8] S. Yadav, S. Prakash, N. Dayal, V. Singh, Forensics Analysis of WhatsApp in Android Mobile Phone, SSRN Electron. J. (2020). doi:10.2139/ssrn.3576379.

[9] S. K. Jha, Whatsapp Deleted Message Forensics a Complete Production-Grade Investigation Framework, 2025.

[10] G. Davies, et al, Security Analysis of the WhatsApp End-to-End Encrypted Backup Protocol, in: Advances in Cryptology — CRYPTO 2023, Springer Nat. Switz., Cham, 2023, pp. 330-361. doi:10.1007/978-3-031-38551-3_11.

[11] k6nmx, k6nmx/WhatsAppDecryption, Software, v. 2440cd4, GitHub, 2014. https://github.com/k6nmx/WhatsAppDecryption.

[12] L. Monk, Decrypting the WhatsApp Crypt8 — A Game of Commands, 2015.

[13] D. Palma, ElDavoo/wa-crypt-tools, Software, v. 765ab62, GitHub, 2026.

[14] M. Dworkin, Recommendation for Block Cipher Modes of Operation:, National Institute of Standards and Technology, Gaithersburg, MD, 2007. doi:10.6028/nist.sp.800-38d.

[15] S. Krassovsky, G. Cadden, How WhatsApp is Enabling End-to-End Encrypted Backups, 2021.

[16] WerWolw, ImHex, Software, v. 1.38.1, WerWolw, 2025.

[17] M. Gracie, et al., DB Browser for SQLite, Software, v. 3.13.99, 2024.

[18] J. Wu, Magisk, Software, v. 28.1.