УДК 004.056.5:004.056.8
DOI: 10.31673/2409-7292.2025.016526

Опірський І.Р., Хохлачова Ю.Є., Стефанків А.В., Шевчук Ю.А.

АНАЛІЗ ТЕХНІЧНИХ ОСОБЛИВОСТЕЙ РЕАЛІЗАЦІЇ ШИФРУВАННЯ ДАНИХ НА SD-КАРТАХ В ANDROID

У статті досліджуються механізми шифрування даних на знімних носіях у операційній системі Android. Проведено детальний аналіз двох основних підходів до захисту інформації: пофайлове шифрування при використанні SD-картки як знімного носія (Portable Storage) та повнодискове шифрування при використанні картки пам'яті як розширення внутрішньої пам'яті пристрою (Adoptable Storage). Досліджено технічні особливості реалізації шкірних методів, включаючи використовувані алгоритми шифрування, структуру зашифрованих даних та механізми зберігання ключів.

В результаті дослідження встановлено, що для повнодискового шифрування використовується модуль ядра dm-crypt у режимі plain із шифром AES-256-CBC-ESSIV:SHA256, а для пофайлового шифрування застосовується модуль ядра eCryptFS. Визначено місця зберігання ключів шифрування та проаналізовано структуру зашифрованих даних для обох методів. Виявлено, що при використанні режиму Adoptable Storage забезпечується більш комплексний захист даних завдяки повнодисковому шифруванню, тоді як режим Portable Storage з пофайловим шифруванням надає більшу гнучкість у використанні, але може бути менш захищеним через можливість аналізу структури файлової системи та метаданих файлів.

Особливу увагу в дослідженні приділено аналізу процесів шифрування та розшифрування даних у кожному з режимів. Встановлено, що в режимі Portable Storage використовується система пофайлового шифрування, яка створює для шкірного файлу унікальний ключ шифрування (File Encryption Key, FEK), який у свою чергу шифрується головним ключем користувача. При цьому зашифровані дані зберігаються разом з метаданими, що містять інформацію про використані алгоритми шифрування та інші параметри. В режимі Adoptable Storage застосовується повнодискове шифрування, при якому створюється єдиний ключ шифрування для всього розділу, що зберігається у захищеній області внутрішньої пам'яті пристрою.

Проведене дослідження також виявило, що реалізація механізмів шифрування може відрізнятись залежно від виробника пристрою та версії операційної системи Android, що створює додаткові складнощі при аналізі безпеки даних на знімних носіях. Зокрема, виявлено відмінності у реалізації механізмів зберігання ключів шифрування та організації структури зашифрованих даних у різних виробників пристроїв.

Окремо розглянуто питання безпеки зберігання ключів шифрування та можливості їх компрометації. Встановлено, що при використанні режиму Adoptable Storage ключі шифрування зберігаються в захищеній області пам'яті пристрою, доступ до якої можливий лише з root-правами. Це забезпечує додатковий рівень захисту, але водночас створює ризики при отриманні зловмисником підвищених привілеїв доступу до системи.

Результати дослідження мають практичне значення для розуміння рівня захищеності даних при використанні різних режимів роботи зі знімними носіями в системі Android та можуть бути використані для удосконалення існуючих механізмів захисту інформації. Отримані дані також можуть бути корисними при розробці рекомендацій щодо безпечного використання знімних носіїв та при проведенні аудиту безпеки мобільних пристроїв.

Ключові слова: Android, шифрування даних, знімні носії, SD-карта, Portable Storage, Adoptable Storage, dm-crypt, eCryptFS, інформаційна безпека, криптографічний захист.

Вступ

З активним розвитком інформаційних технологій та повсякденним використанням мобільних пристроїв, обсяги даних, що зберігаються та обробляються на смартфонах, планшетах та інших портативних пристроях, зростають експоненційно. Значна частина цієї інформації має конфіденційний характер: особисті фотографії, документи, паролі, фінансові дані та інша чутлива інформація. Це робить питання захисту даних на мобільних пристроях одним із ключових аспектів інформаційної безпеки у сучасному цифровому світі.

Операційна система Android, яка займає лідируючі позиції на ринку мобільних платформ, надає користувачам різноманітні механізми захисту їх даних. Особливе місце серед них посідають технології шифрування інформації, що зберігається як у внутрішній пам'яті пристроїв, так і на знімних носіях. З огляду на те, що багато користувачів використовують SD-картки для розширення пам'яті своїх пристроїв, питання захисту даних на знімних носіях набуває особливої актуальності.

В операційній системі Android реалізовано два принципово різні підходи до роботи зі знімними носіями: режим портативного накопичувача (Portable Storage) та режим адаптивного сховища (Adoptable Storage). Кожен з цих режимів має свої особливості реалізації механізмів шифрування даних, що безпосередньо впливає на рівень захищеності інформації користувачів. При цьому важливо розуміти не лише загальні принципи роботи цих механізмів, а й технічні деталі їх реалізації, включаючи використані алгоритми шифрування, методи зберігання ключів та особливості організації зашифрованих даних.

Актуальність дослідження механізмів шифрування даних на знімних носіях у системі Android обумовлена також тим, що розуміння технічних аспектів їх реалізації є критично важливим для оцінки реального рівня захищеності інформації та потенційних уразливостей. Це особливо важливо в контексті зростання кількості кіберзагроз та постійного вдосконалення методів несанкціонованого доступу до даних. Глибоке розуміння принципів роботи систем шифрування дозволяє не лише оцінювати їх ефективність, а й розробляти рекомендації щодо їх удосконалення та безпечного використання.

Варто зазначити, що дослідження механізмів шифрування в Android має важливе значення не лише для розробників та фахівців з інформаційної безпеки, а й для звичайних користувачів, яким важливо розуміти, наскільки надійно захищені їх дані при використанні різних режимів роботи зі знімними носіями. Це розуміння дозволяє приймати обґрунтовані рішення щодо вибору методів зберігання та захисту важливої інформації.

Постановка проблеми

В умовах швидкого розвитку мобільних технологій та зростання вимог до захисту конфіденційних даних, питання дослідження механізмів шифрування інформації на мобільних пристроях набуває особливої значущості. Незважаючи на те, що операційна система Android надає вбудовані засоби шифрування даних на знімних носіях, технічні аспекти їх реалізації залишаються недостатньо дослідженими.

Основна проблема полягає у відсутності систематизованих знань про особливості реалізації різних методів шифрування даних при використанні SD-карток в системі Android. Зокрема, потребують детального вивчення механізмів шифрування в режимах Portable Storage та Adoptable Storage, включаючи аналіз використовуваних алгоритмів, структури зашифрованих даних та методів зберігання ключів шифрування.

Додаткову складність створює те, що різні виробники мобільних пристроїв можуть використовувати власні модифікації стандартних механізмів шифрування, що ускладнює розуміння загальної картини захищеності даних на знімних носіях. Крім того, відсутність документації щодо деяких аспектів реалізації шифрування ускладнює процес дослідження та оцінки ефективності цих механізмів.

Вирішення цієї проблеми вимагає проведення комплексного технічного аналізу механізмів шифрування даних на знімних носіях в операційній системі Android, що дозволить краще зрозуміти рівень захищеності інформації та виявити потенційні вразливості у існуючих методах захисту даних.

Метою статті є аналіз технічних особливостей реалізації механізмів шифрування даних на знімних носіях в операційній системі Android, визначення використовуваних алгоритмів шифрування та оцінка їх ефективності.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Дослідити особливості реалізації пофайлового шифрування при використанні SD-картки в режимі Portable Storage.
2. Проаналізувати механізми повнодискового шифрування при використанні SD-картки в режимі Adoptable Storage.
3. Визначити алгоритми шифрування та методи зберігання ключів, що використовуються в обох режимах.
4. Оцінити ефективність та надійність досліджуваних механізмів шифрування.

Аналіз останніх досліджень та публікацій

В операційній системі Android існують три варіанти підключення SD-картки як носія даних: знімний носій (Portable Storage) [1], розширення внутрішнього сховища (Adoptable Storage) [2] та змішаний режим (Mixed Storage або Semi-Adoptable Storage) [3].

При підключенні картки як знімного носія на ній створюється файлова система FAT32 або exFAT (за їх відсутності), після чого файлова система монтується за допомогою /storage/XXXX-XXXX, де XXXX-XXXX – серійний номер файлової системи. За такого під'єднання на карту пам'яті можливе збереження лише користувацьких файлів, збереження застосунків неможливе через обмеження файлових систем FAT32 та exFAT порівняно з файловими системами f2fs [4] та ext4 [5], що використовують для внутрішнього сховища пристрою. При цьому підключення картки саме як зовнішнього сховища залишається єдиним методом на переважній більшості пристроїв, у яких все ще є слот для SD-карт. Цей метод підключення є оптимальним для переважної більшості користувачів, оскільки таку SD-карту можна вилучити з пристрою та підключити до комп'ютера для перенесення даних.

При підключенні картки як розширення внутрішнього сховища на ній створюється розділ із тією файловою системою, яка використовується для розділу /data самого пристрою. Створена файлова система об'єднується з файловою системою на пристрої за принципом, подібним до створення складеного в ОС Windows. За такого підключення картки на неї можливе перенесення не лише користувацьких даних, а й застосунків, системної інформації тощо. Використання файлових систем сімейства FAT та NTFS для зберігання даних застосунків неможливе, оскільки перелічені файлові системи не підтримують атрибути файлів, які використовують операційні системи на основі ядра Linux, зокрема Android. Такий метод підключення не набув популярності через його незручність, а також через те, що не всі виробники пристроїв дозволяли використовувати цей метод. Наразі використання Adoptable Storage є недоцільним у зв'язку зі зростанням обсягу внутрішнього сховища пристроїв до прийнятних значень, а також вилученням слоту для SD-карт із нових устройств. Одними з головних переваг режиму Adoptable Storage були безшовне розширення внутрішньої пам'яті та надійне шифрування даних.

Також в ОС Android 6.0 у деяких виробників (наприклад, Motorola) була доступна опція Mixed Storage. Гібридний режим був спробою знайти компромісне рішення між повністю портативним та повністю адаптивним режимами використання SD-карт в Android. Розглянемо детально його реалізацію та причини обмеженого поширення.

Semi-Adoptable Storage передбачає розділення SD-карти на два логічні розділи. Перший розділ працює як адаптивне сховище (Adoptable Storage), інтегруючись із внутрішньою пам'яттю пристрою та дозволяючи встановлювати застосунки. Інший розділ функціонує як звичайний портативний накопичувач, доступний для зберігання та перенесення файлів між пристроями.

При налаштуванні гібридного режиму система виконує наступні дії:

1. Створює на карті два розділи з різними файловими системами: ext4 для адаптивної частини та FAT32/exFAT для портативної.
2. Шифрує адаптивний розділ, інтегруючи його із системною файловою структурою Android.
3. Встановлює окремі правила доступу та керування даними для кожного розділу.
4. Налаштовує системні служби для одночасної роботи з обома типами сховища.

Цей режим не став популярним і був швидко забутий виробниками та розробниками з низки причин:

- Технічна складність: Підтримка двох різних режимів роботи на одній карті пам'яті створює додаткове навантаження на систему та ускладнює керування пам'яттю. Це може призводити до зниження продуктивності та збільшення споживання енергії.

● Складність для користувачів: концепція розділеного сховища виявилася надто складною для розуміння звичайними користувачами. Багатьом було важко визначити, які дані зберігаються в якомусь розділі, що призводило до плутанини.

● Проблеми з надійністю: при використанні гібридного режиму зростає ризик пошкодження даних через більш складну структуру файлової системи та більше операцій з даними.

● Обмежена підтримка виробників: Більшість виробників смартфонів надали перевагу реалізації найпростіших і зрозуміліших режимів роботи з SD-картами, що призвело до обмеженої підтримки гібридного режиму на рівні пристроїв.

● Альтернативні рішення: з появою пристроїв із великим обсягом внутрішньої пам'яті та розвитком хмарних сховищ потреба у складних режимах роботи з SD-картами зменшилася.

Незважаючи на те, що гібридний режим пропонував цікаве технічне рішення, складність його реалізації та використання, разом зі змінами в потребах користувачів, призвели до того, що він не набув широкого поширення в екосистемі Android.

Результати дослідження

Шифрування даних при підключенні картки як знімного носія

Перевірка використання шифрування даних у режимі підключення Portable Storage проводилася з використанням пристрою Xiomi Redmi 12 (Android 13.0.17, офіційна прошивка, версія 12 SKQ1.210908.001).

Тестова SD-карта була відформатована засобами системи Android та підключена як зовнішнє сховище. Під час копіювання даних вони були збережені у вихідному вигляді без шифрування. При включенні опції шифрування (рис. 1) файли, які вже були на карті SD, зашифровані не були. Після ввімкнення шифрування даних на картку пам'яті було скопійовано 9 фотографій загальним розміром 55,6 мегабайт. Процес копіювання зайняв приблизно 25 секунд. Такий результат вказує на значне обмеження швидкості введення-виведення для операцій із шифруванням даних. Це може бути викликано як недостатніми апаратними можливостями пристрою, так і особливостями роботи конкретної версії ПЗ, що працює з досить великими затримками та нагріванням пристрою. Після копіювання фотографії залишилися доступними для перегляду засобами пристрою, а також при підключенні пристрою до ПК у режимі MTP. При підключенні карти пам'яті безпосередньо до ПК файли зберігають свої імена, розширення, але не зберігають метадані та стають недоступними для перегляду та редагування.

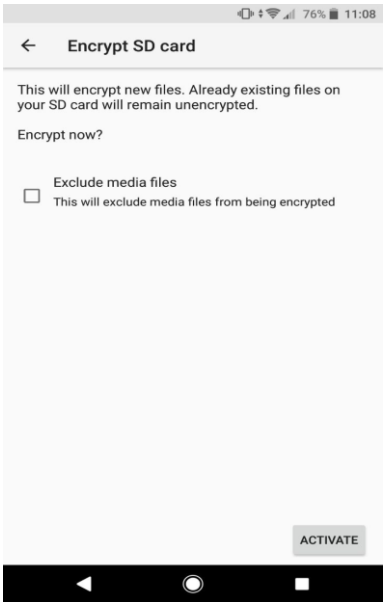

Рис. 1. Увімкнення опції шифрування файлів

© Опірський І.Р., Хохлачова Ю.Є., Стефанків А.В., Шевчук Ю.А. (2025). Аналіз технічних особливостей реалізації шифрування даних на SD-картах в Android. Сучасний захист інформації, 1(61), 219–228. https://doi.org/10.31673/2409-7292.2025.016526.

При аналізі структури зашифрованого файлу за допомогою шістнадцяткового редактора було виявлено заголовок, який відповідає формату шифрування пофайлового утиліти eCryptFS [6] (рис. 2).

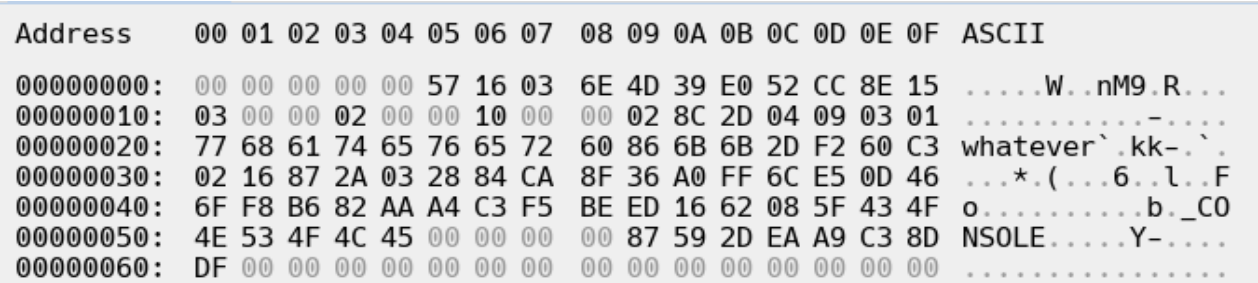

Рис. 2. Заголовок файлу, зашифрованого пристроєм

Заголовок файлу складається з інформації про вихідний розмір файлу, маркер формату, прапори, кількість екстентів (безперервних сегментів даних, сегментація необхідна для забезпечення довільного доступу до даних), їх розмір (відповідає розміру блоку файлової системи), зашифрований ключ шифрування файлу і, власне, самі екстенти (починаючи зі зміщення 0x). Для шифрування екстентів використовується алгоритм AES в режимі зчеплення блоків шифротексту (CBC) з довжиною ключа 128 біт.

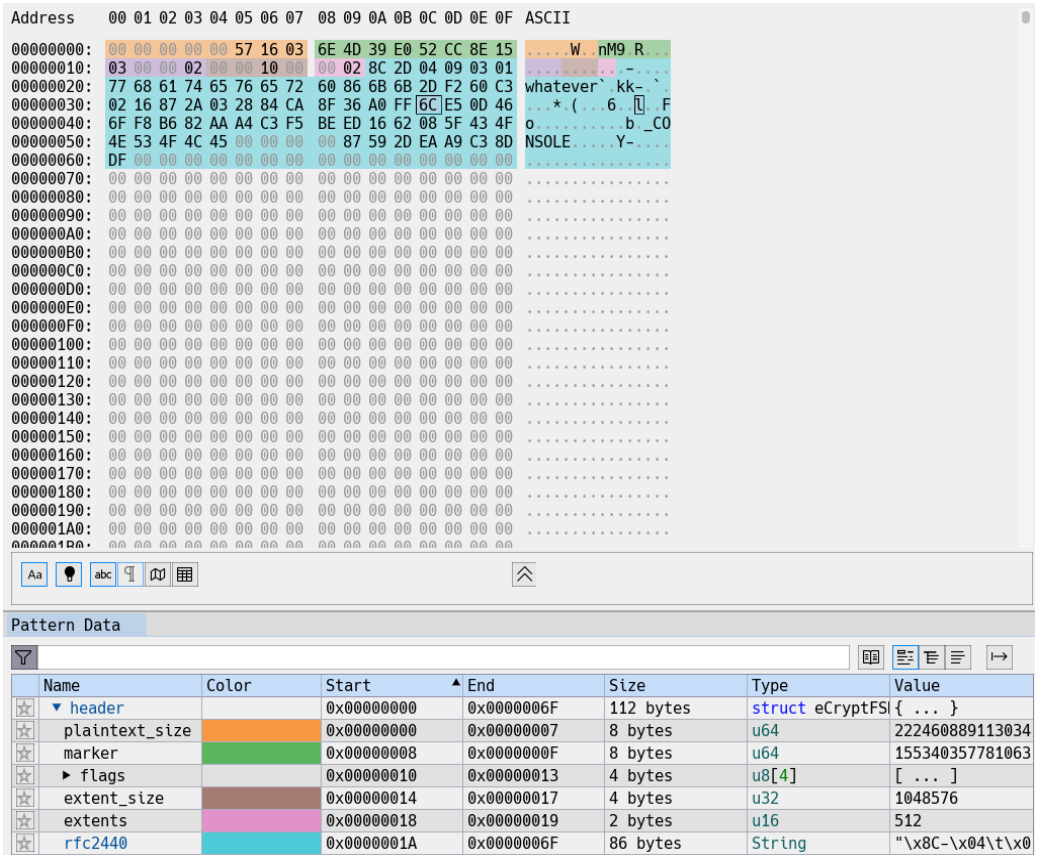

Рис. 3. Структура заголовка файлу eCryptFS

У процесі шифрування та розшифрування файлів за допомогою eCryptFS задіяно декілька основних етапів та компонентів (Рис. 4).

© Опірський І.Р., Хохлачова Ю.Є., Стефанків А.В., Шевчук Ю.А. (2025). Аналіз технічних особливостей реалізації шифрування даних на SD-картах в Android. Сучасний захист інформації, 1(61), 219–228. https://doi.org/10.31673/2409-7292.2025.016526.

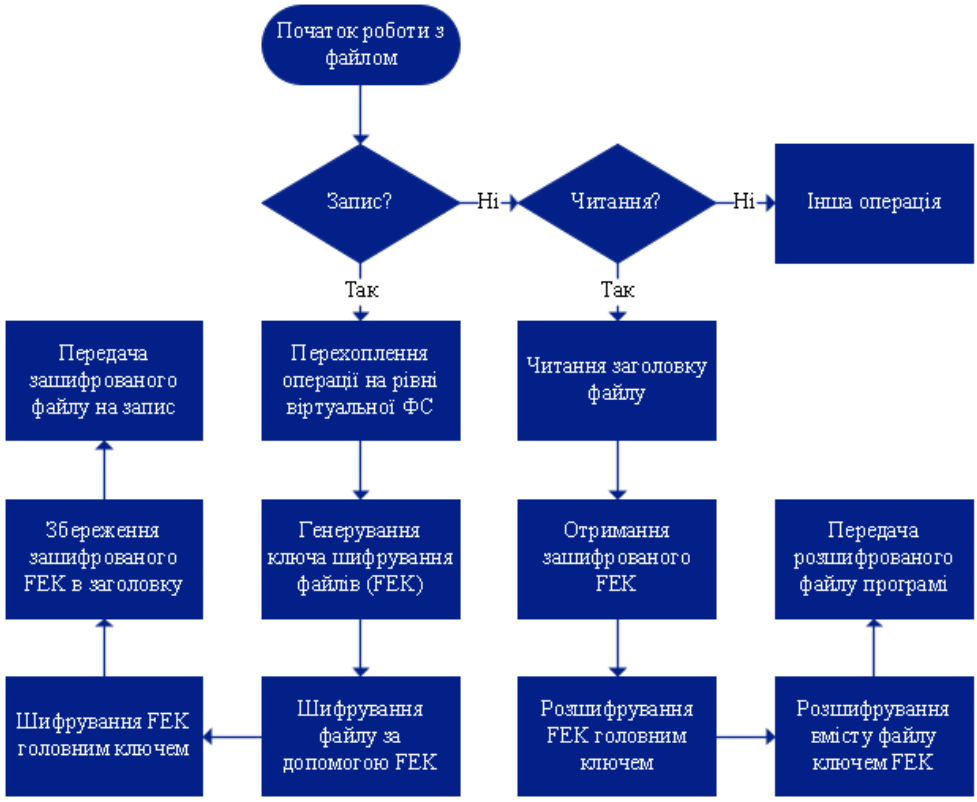

Рис. 4. Процес роботи з файлом eCryptFS

Коли користувач створює або зберігає файл у зашифрованому каталозі, eCryptFS спочатку перехоплює операцію на рівні віртуальної файлової системи (VFS). На цьому етапі система генерує унікальний ключ файлу шифрування (File Encryption Key, FEK) [8]. Цей ключ створюється випадково для кожного окремого файлу.

Далі система використовує згенерований FEK для шифрування вмісту файлу. Шифрування відбувається посторінково, зазвичай блоками по 4 КБ. Це забезпечує ефективний доступ до окремих частин файлу без необхідності розшифровувати весь файл. Сам FEK шифрується за допомогою головного ключа користувача, отриманого з пароля користувача.

Зашифрований FEK зберігається в метаданих файлу разом з додатковою інформацією, такою як використаний алгоритм шифрування та інші параметри. Після цього зашифровані дані та метадані передаються до нижнього рівня файлової системи для фізичного зберігання на диску.

Коли користувач намагається отримати доступ до зашифрованого файлу, процес відбувається у зворотному порядку. Спочатку система зчитує метадані файлу та витягує зашифрований FEK. Використовуючи головний ключ користувача (який отриманий з пароля під час монтування файлової системи), система розшифровує FEK.

Після отримання розшифрованого FEK, система використовує його для розшифрування вмісту файлу. Розшифрування також відбувається посторінково, що дозволяє ефективно працювати з великими файлами. Розшифровані дані передаються до програми, яка запросила доступ до файлу.

eCryptFS забезпечує прозоре шифрування, що означає автоматичне шифрування та розшифрування файлів без необхідності спеціальних дій з боку користувача чи програм. Система також підтримує шифрування файлів, що забезпечує додатковий рівень конфіденційності.

Важливо, що eCryptFS зберігає структуру каталогів незашифрованою, шифруючи лише вміст файлів і, за необхідності, їх імена. Це дозволяє файловій системі ефективно працювати з деревом каталогів, зберігаючи при цьому конфіденційність даних.

Щоб розшифрувати файл, зашифрований eCryptFS, необхідно отримати FEK. У повноцінних операційних системах сімейства GNU/Linux файл із зашифрованим ключем монтування (майстер-ключом), який знаходиться в домашній директорії користувача та розшифровується або за допомогою парольної фрази/ключа (легітимний метод), або за допомогою перебору (нелегітимний метод). Оскільки в ОС Android немає директорії користувача, то наступним кроком буде перевірка директорії сховища ключів /data/misc/keystore/user_0 (зауважимо, що ця директорія доступна тільки з root-правами). В результаті перебору всіх файлів ключів у цій директорії за допомогою утиліти ecryptfs2john зі стандартного набору дистрибутива Kali Linux жоден ключ не був розпізнаний як wrapped-passphrase утиліти eCryptFS (рис. 5).

```
./10047_USRPKEY_Stats_Key_EC:$ecryptfs$0$
./10047_USRPKEY_c5635988b7157a3a:$ecryptfs$0$
./10047_USRPKEY_nearby_sharing_paired_key_alias_2KPUzcGZ9_7MJy383crVh4DBRij1TqObMgMKYVEToD0:$ecryptfs$0$
./10047_USRPKEY_nearby_sharing_paired_key_alias_5A5T8C4seHP+]MoaLuroUvGsMbxpE4ODZZaFAUvGmUbc:$ecryptfs$0$
./10047_USRPKEY_nearby_sharing_paired_key_alias_6GnRExrR5fhyqnPvnbqImiGHlYkL3YKOJBgA5HvU_A4:$ecryptfs$0$
./10047_USRPKEY_nearby_sharing_paired_key_alias_FNjb278H55+]s0wy9Q5bkPhsdKdQsDbyr5+]azv9jcaig:$ecryptfs$0$
./10047_USRPKEY_nearby_sharing_paired_key_alias_Q23NUZA4Jbzqwji7h_VctUHEtuNX32wsljnecEMRjTg:$ecryptfs$0$
./10047_USRPKEY_nearby_sharing_paired_key_alias_RLkrRBcGk2FdpqVyBHJZrwv7lQCux_7is_8d17D4hU8:$ecryptfs$0$
./10047_USRPKEY_nearby_sharing_paired_key_alias__WkyPTF1_pbsvoaTSv80ZwlZfZ44YPwd4hkeNq5JSPQ:$ecryptfs$0$
./10047_USRPKEY_nearby_sharing_paired_key_alias_dy+]YPvTVUDkkXkNW6cq4rtN3S8yB40Awnxt6vdEqeKE:$ecryptfs$0$
./10047_USRPKEY_nearby_sharing_paired_key_alias_omcBF1a3C7ooH+]_RlfQDz43WPBVXgUtl9GGcnV4WOMQ:$ecryptfs$0$
./10047_USRSKEY__androidx_security_master_key_:$ecryptfs$0$
./10047_USRSKEY_mobstore_encrypt:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_1TLaHtFvFMpv42O57LStFLhEcAwqeQii2SfRuH_eGdc:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_3h65piZeXuDTFPw0fuKh+]RJx7fky8ln5T9GJV8IVaXQ:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_4LSpxyk5H80sgRnwun3eE+]9cYth5BGzfdUkoBZTQS8s:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_4Xzdg_anXWWX7H4k3Tl1lIz6KoSDIKd3zsGeQHUQdGk:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_6LHjUE9zahbmHnpSCb6E3pPukQDbmqVU__O3Wy2RuqU:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_9IDIu8aTJXi0rOaDruieKqR6EGlQX0khfggjAgGNxL8:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_99a97dbNeB9CfFovRB44LTIdrbFD_aVj5KlXUHN6hLQ:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_Al_QgXIMu_9loA9AG6iu1s5Ebd3Cq9oPC88u2LiHIWk:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_PUdBYdwt6Jsy+]bQbW44EvYaWKrO4ZE7OMQlH58T9hyE:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_T80S4pnm21CxJnrarSwem30JLS3Etqm+]YYxWe2bRwf0:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_TyLB_P1RjYeq8k0F7UVq2LZmiBfA1RTevTJcRloLNf4:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_XS_8p7O9wFQCTmGUzPiccrCgziCSqAQoka_bWpjL1sY:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_gUYZLTUdDhxOOqmSyVPt_VqhzT6mZ9ssph8zh01eb3k:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_qQAHhOVWGZGN9VbGLfNFDSIAJi3vJB2t+]bunbDSZVc8:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_rkIYcPmoOBijegJxLnfWcYqEza3DvgCs2FXgAR+]5C7U:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_rnRvxhEejNnZXelgBbWtlQjMU2+]PdOwhDQ34GXxW_VY:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_rvVAsfyhKOW+]YWp34MVap6l0mORDBsy0i8qpWLdoa00:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_t_hThpFRX9wv4eRssl0Xs3O98WyaknlpzaKPPWUh4rA:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_xLFB0Y2y2OV4p28bhYJ_WDRMHX0_Jh6NBdDPqWV8Voo:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_xg65BhktRkqMvF8IeDlwbrfrMnwlSfMEr6ROxIpDvYY:$ecryptfs$0$
./10047_USRSKEY_nearby_presence_connection_secret_key_alias_zbuzMcZmXDidINst03OSIh+]fR3FN4ARNy_62Y8514Oc:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000003_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000004_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000005_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000006_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000007_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000008_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000009_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000010_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000012_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000013_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000014_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000015_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000016_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000017_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_1000018_1:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_5000001_2:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_5000001_3:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_5000002_2:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_8000001_4:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_8000001_5:$ecryptfs$0$
./10059_USRSKEY_gmphn_model_key_8000001_6:$ecryptfs$0$
./10059_USRSKEY_gmphn_pkey:$ecryptfs$0$
./10111_USRSKEY__androidx_security_master_key_:$ecryptfs$0$
./10216_USRCERT_tmessages_passcode:$ecryptfs$0$
./10217_USRSKEY_bg_token:$ecryptfs$0$
./10224_USRSKEY_org+^mozilla+^fennec_fdroid:$ecryptfs$0$
```

Рис. 5. Перевірка наявності ключів шифрування eCryptFS у сховищі ключів системи Android

Використання eCryptFS є специфічним для кількох виробників пристроїв на основі Android, серед них найбільш відомими є Sony та Samsung. Утиліти для роботи з eCryptFS у прошивках цих виробників не є стандартними та доступними через консоль суперкористувача,

тому подальші кроки мають на увазі пофайлову перевірку файлової системи та аналіз бінарних файлів прошивки з метою виявлення місць збереження ключів розшифрування файлів.

Шифрування даних при підключенні картки як розширення внутрішньої пам'яті

Для перевірки шифрування Adoptable Storage використовувався Pixel 4a (Android 13.0.15, прошивка стороння, версія PixelExperience 13). Карта була відформатована для Adopted Storage за допомогою стандартних засобів прошивки та підключена з переміщенням 104 мегабайт даних із внутрішньої пам'яті. Після переміщення даних карту SD було вилучено з попереднім вимкненням пристрою для подальшого аналізу.

Штатні засоби операційної системи створили два розділи на карті пам'яті (рис. 6) з невідомою файловою системою.

```
gilah@magi-arch ~ % sudo fdisk -l /dev/mmcblk0
Disk /dev/mmcblk0: 29.12 GiB, 31266439168 bytes, 61067264 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: B7B39FFC-D913-4603-A07E-725D555A5F32

Device         Start      End  Sectors  Size Type
/dev/mmcblk0p1  2048    34815    32768   16M unknown
/dev/mmcblk0p2 34816 61067230 61032415 29.1G unknown
```

Рис. 6. Структура розділів картки пам'яті в режимі Adoptable Storage

Перевірка за допомогою вбудованих утиліт cryptsetup [9] та dmsetup [10] (рис. 7) також не виявила використання LUKS, з чого можна зробити висновок, що використовується режим plain модуля dm-crypt, який не містить заголовків розділів і фактично не відрізняється від випадкових даних.

```
gilah@magi-arch ~ % sudo blkid /dev/mmcblk0p1
/dev/mmcblk0p1: UUID="4936-1C14" BLOCK_SIZE="512" TYPE="vfat" PARTLABEL="android_meta" PARTUUID="c6fb2eca
gilah@magi-arch ~ % sudo blkid /dev/mmcblk0p2
/dev/mmcblk0p2: PARTLABEL="android_expand" PARTUUID="419aa35f-5fcb-4ea7-8b6b-c968eeed90eb"
gilah@magi-arch ~ % sudo cryptsetup luksDump /dev/mmcblk0p1
Device /dev/mmcblk0p1 is not a valid LUKS device.
1 gilah@magi-arch ~ % sudo cryptsetup luksDump /dev/mmcblk0p2
Device /dev/mmcblk0p2 is not a valid LUKS device.
1 gilah@magi-arch ~ %
```

Рис. 7. Результат перевірки щодо використання технології LUKS

Dm-crypt підтримує шифрування як з використанням звичайного режиму (plain mode), так і з використанням LUKS (Linux Unified Key Setup). Режим plain є базовим режимом шифрування, який надає лише основну функціональність шифрування без додаткових можливостей керування ключами. При використанні plain режиму пароль безпосередньо перетворюється на ключ шифрування через хеш-функцію. Цей режим не зберігає жодних метаданих на диску, що робить його стійкішим до криптоаналізу, але значно менш зручним у використанні. LUKS є більш сучасним та функціональним рішенням. Він створює спеціальний заголовок на диску, який містить всю необхідну інформацію про параметри шифрування та ключі. LUKS підтримує до восьми різних ключів для одного розділу, що дозволяє надавати доступ різним користувачам або мати резервні ключі. Важливою особливістю є використання майстер-ключа, який фактично шифрує дані, тоді як паролі користувачів шифрують цей майстер-ключ.

Що стосується практичного застосування, LUKS є рекомендованим варіантом для більшості випадків використання, оскільки він забезпечує:

© Опірський І.Р., Хохлачова Ю.Є., Стефанків А.В., Шевчук Ю.А. (2025). Аналіз технічних особливостей реалізації шифрування даних на SD-картах в Android. Сучасний захист інформації, 1(61), 219–228. https://doi.org/10.31673/2409-7292.2025.016526.

- Стандартизований формат заголовка на диску;
- Можливість зміни паролів без повторного шифрування даних;
- Поліпшений захист від атак перебором завдяки використанню сольових значень;
- Підтримка керування кількома ключами.

Режим plain може бути корисним у специфічних сценаріях, наприклад:

- Коли потрібна максимальна скритність самого факту шифрування;
- створення власних систем управління ключами;
- У випадках коли критично важлива простота реалізації.

Варто зазначити, що LUKS вимагає додаткового місця на диску для зберігання заголовка (зазвичай кілька мегабайт), тоді як режим plain не вимагає додаткового простору. Однак ця різниця є несуттєвою для сучасних систем зберігання даних.

В операційній системі Android ключ шифрування розділу Adoptable Storage зберігається шляхом /data/misc/vold. Розділ може бути розшифрований у разі отримання ключа шифрування за допомогою root-прав або інших методів [11]. Файл ключа було успішно перенесено на комп'ютер, після чого розділ карти пам'яті /dev/mmcblk0p2 був успішно розшифрований. Методом підбору було з'ясовано, що використовується алгоритм шифрування AES-128 як CBC з використанням алгоритму обчислення векторів ініціалізації ESSIV (рис. 8).

```
gilah@magi-arch ~ % sudo cryptsetup open --type plain --cipher aes-cbc-essiv:sha256 \
\`> --key-file expand_419aa35f5fcb4ea78b6bc968eeed90eb.key --key-size=128 /dev/mmcblk0p2 adoptable

gilah@magi-arch ~ % sudo mount /dev/mapper/adoptable /mnt
gilah@magi-arch ~ % ls -lha /mnt
total 30K
drwxr-xr-x  7 root  root  4.0K Dec  2 22:51 .
drwxr-xr-x 19 root  root  4.0K Nov 29 16:18 ..
drwxrwx--x  2 gilah gilah 3.5K Dec  2 22:51 app
drwxr-x--x  3 root  root  3.5K Dec  2 22:51 local
drwxrwx---  3  1023  1023 3.5K Dec  2 22:51 media
drwx--x--x  3 gilah gilah 3.5K Dec  2 22:51 user
drwx--x--x  3 gilah gilah 3.5K Dec  2 22:51 user_de
gilah@magi-arch ~ % sudo blkid /dev/mapper/adoptable
/dev/mapper/adoptable: UUID="7c888bdc-c501-4f58-8b9b-6c07734eff87" BLOCK_SIZE="4096" TYPE="f2fs"
gilah@magi-arch ~ %
```

Рис. 8. Успішне розшифрування розділу SD-карти та його вміст

Оскільки картка пам'яті була щойно відформатована, на ній не знаходиться жодних файлів, тільки створена системою структура директорій (Рисунок. 9). Призначення директорій відповідає їхній назві.

```
gilah@magi-arch ~ % sudo tree /mnt
/mnt
|-- app
|-- local
|   `-- tmp
|-- media
|   `-- 0
|-- user
|   `-- 0
`-- user_de
    `-- 0

10 directories, 0 files
gilah@magi-arch ~ %
```

Рис. 9. Структура незаповненої картки пам'яті в режимі Adoptable Storage

Аналіз розшифрованих даних показав, що система створює стандартну структуру директорій на карті пам'яті, яка відповідає основним категоріям даних, що можуть зберігатися на пристрої. При цьому всі файли, які будуть записані на картку пам'яті, автоматично шифруються системою за допомогою того ж ключа шифрування.

Важливо відзначити, що при вилученні картки пам'яті з пристрою доступ до даних стає неможливим без наявності відповідного ключа шифрування. Це забезпечує надійний захист інформації у разі втрати чи викрадання картки пам'яті. Однак, як показало дослідження, наявність root-доступу до пристрою дозволяє отримати ключ шифрування та, відповідно, доступ до даних на карті пам'яті. Порівнюючи обидва досліджені методи шифрування даних, можна зазначити, що режим Adoptable Storage забезпечує більш комплексний захист даних завдяки повнодисковому шифруванню, в той час як режим Portable Storage з пофайловим шифруванням надає більшу гнучкість у використанні, але може бути менш захищеним через можливість аналізу структури файлової системи та методу.

Проведене дослідження також виявило, що реалізація механізмів шифрування може відрізнятись залежно від виробника пристрою та версії операційної системи Android, що створює додаткові складнощі при аналізі безпеки даних на знімних носіях. Це підкреслює необхідність подальших досліджень у цьому напрямку, особливо в контексті нових версій Android та різних реалізацій від виробників пристроїв.

Висновки

У цьому дослідженні було розглянуто основні методи шифрування даних на знімних носіях, які використовуються в операційній системі Android: шифрування пофайла і повнодискове шифрування. Пофайлове шифрування може використовуватися при підключенні картки пам'яті як зовнішнього накопичувача, в той час як шифрування повнодискове є обов'язковим при підключенні карти пам'яті як розширення внутрішньої пам'яті пристрою. Було з'ясовано, що повнодискове шифрування використовує модуль ядра dm-crypt в режимі plain з шифром AES-256-CBC-ESSIV: SHA256, ключ шифрування розділу зберігається у внутрішній пам'яті пристрою /data/misc/vold/expand_<PARTUUID>.key; Пофайлове шифрування в досліджуваному пристрої використовує модуль ядра eCryptFS, розташування майстер-ключа якого з'ясувати не вдалося.

Перелік посилань

1. Android Open Source Project. Traditional storage | Android Open Source Project. Android Open Source Project. URL: https://source.android.com/docs/core/storage/traditional (date of access: 27.12.2024).

2. Android Open Source Project. Adoptable storage | Android Open Source Project. Android Open Source Project. URL: https://source.android.com/docs/core/storage/adoptable (date of access: 27.12.2024).

3. What happened to Android's adopted storage option що ви можете зробити SD card як internal storage space? My S10 Plus мав upgrade до Android 12 і я не можу це зробити. Quora. URL: https://www.quora.com/What-happened-to-Android-s-adopted-storage-option-that-allowed-you-to-mount-the-SD-card-as-internal-storage-space-My-S10-Plus-had-an-upgrade-to-Android-2-wh 12.2024).

4. Linux Kernel Organization, Inc. WHAT IS Flash-Friendly File System (F2FS)? - The Linux Kernel documentation. Linux Kernel Documentation – Linux Kernel Documentation. URL: https://docs.kernel.org/filesystems/f2fs.html (date of access: 29.12.2024).

5. Linux Kernel Organization, Inc. ext4 Data Structures and Algorithms – Linux Kernel documentation. Linux Kernel Documentation – Linux Kernel Documentation. URL: https://docs.kernel.org/filesystems/ext4/index.html (date of access: 29.12.2024).

6. Kirkland D. eCryptfs. eCryptfs. URL: https://www.ecryptfs.org/ (date of access: 29.12.2024).

7. Euresys sa eCryptfs Header. Euresys Documentation. URL: https://documentation.euresys.com/Products/PICOLO_NET_HD1/PICOLO_NET_HD1/en-us/Content/encrypted-media-storage/ecryptfs-header.htm?TocPath=Resources|Encrypted%20Media%20Storage|_____2.

8. Linux Kernel Organization, Inc. Натиснуті клавіші для файлу eCryptfs - The Linux Kernel documentation. Linux Kernel Archives. URL: https://www.kernel.org/doc/html/v4.17/security/keys/ecryptfs.html (date of access: 29.12.2024).

9. Cryptsetup/cryptsetup GitLab. GitLab. URL: https://gitlab.com/cryptsetup/cryptsetup (date of access: 29. 12.2024).

10. Kerrisk M. dmsetup(8) - Linux manual page. Michael Kerrisk – man7.org. URL: https://man7.org/linux/man-pages/man8/dmsetup.8.html (date of access: 29.12.2024).

11. POQDavid. How to decrypt and split adopted storage?. XDA Developers URL: https://xdaforums.com/t/how-to-decrypt-and-split-adopted-storage.3383666/ (date of access: 29.12.2024).

Надійшла 18.02.2025