

세션 데이터베이스 복호화 및 주요 아티팩트 분류

2조

20232106 정유진 / 20212052 이동훈

alice2002@kookmin.ac.kr / dhsama51@kookmin.ac.kr

25.05.15

목차

1. 환경 세팅
2. 아티팩트 쌓기 / 추출
3. 아티팩트 분석
4. 어플리케이션 분석 - 기초 조사
5. 어플리케이션 분석 - Android Keystore에서 키 추출
6. 어플리케이션 분석 - DB 암호화 흐름
7. 어플리케이션 분석 - DB 복호화 흐름
8. 어플리케이션 분석 - DB 생성, open, close, 조작 흐름
9. 어플리케이션 분석 정리 / 최종 복호화
10. 아티팩트 최종 분석

환경 세팅

Nox 가상환경에서 분석

Android 12 기준

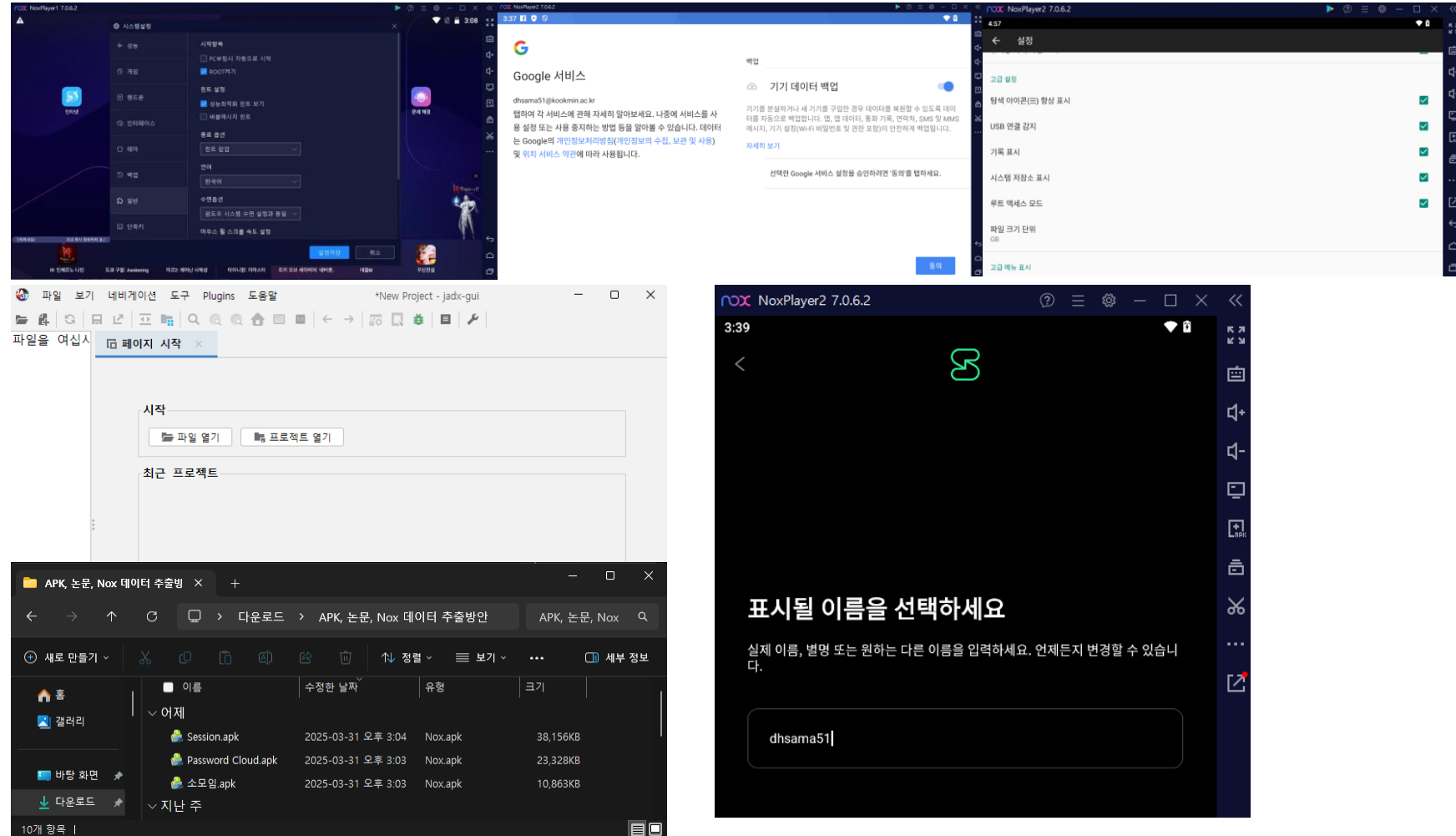
Nox, jadx, Python 사용

Nox - Root, 백업 설정

파일 탐색기 설정

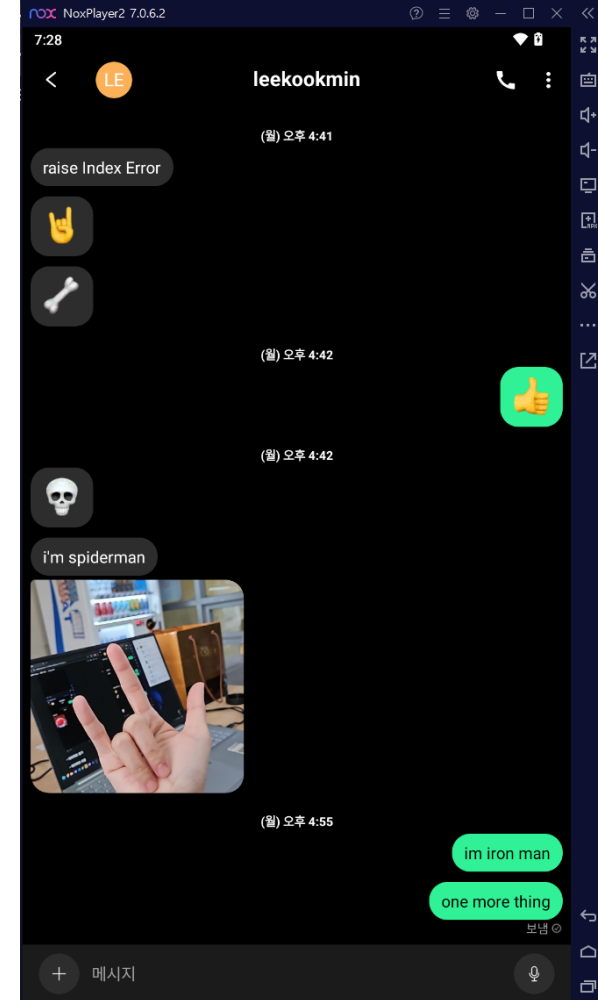
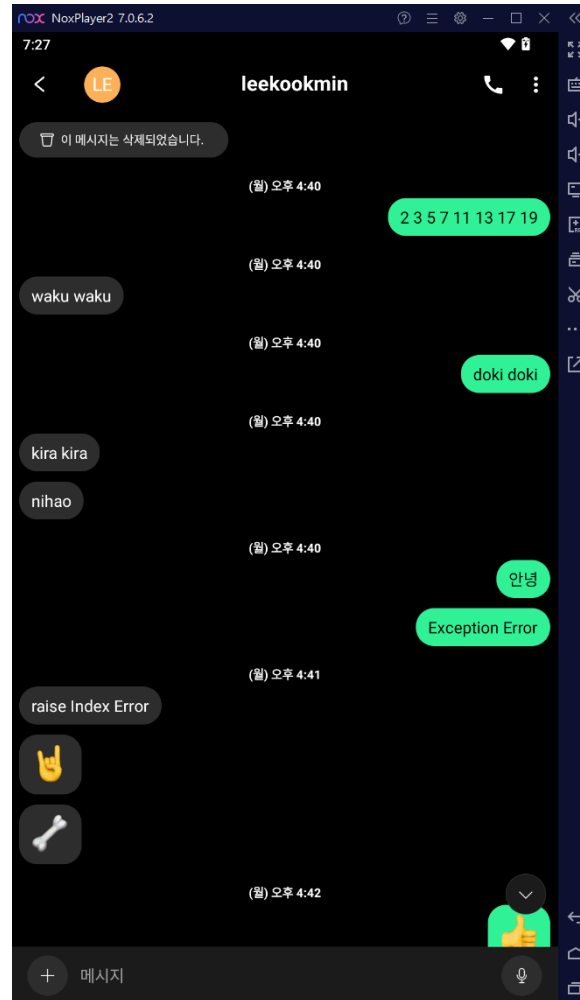
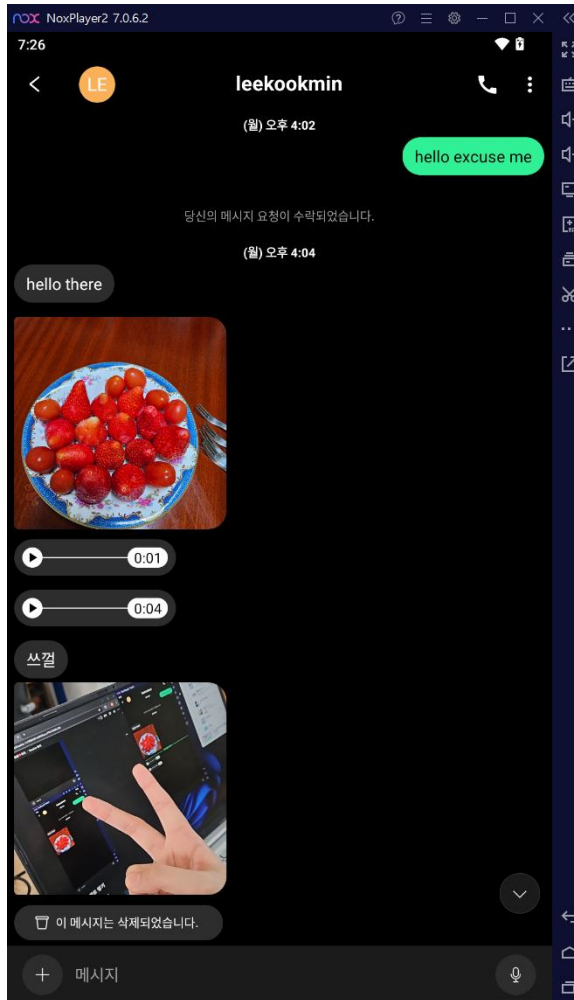
jadx - Session.apk 준비

Session 회원가입



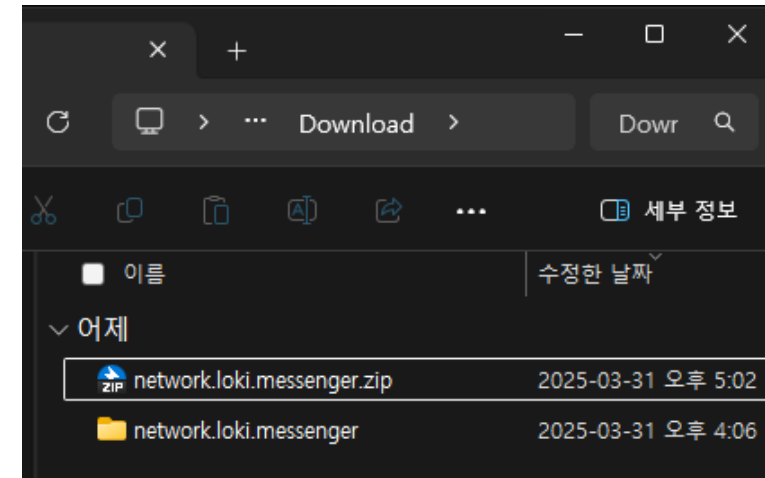
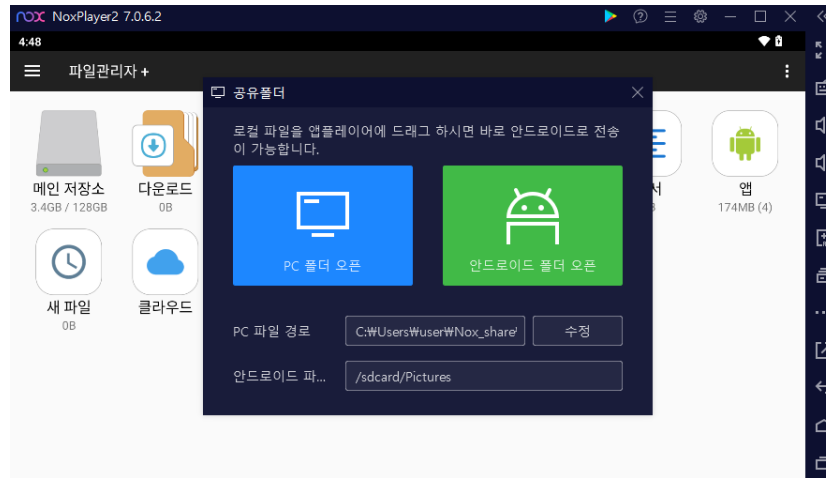
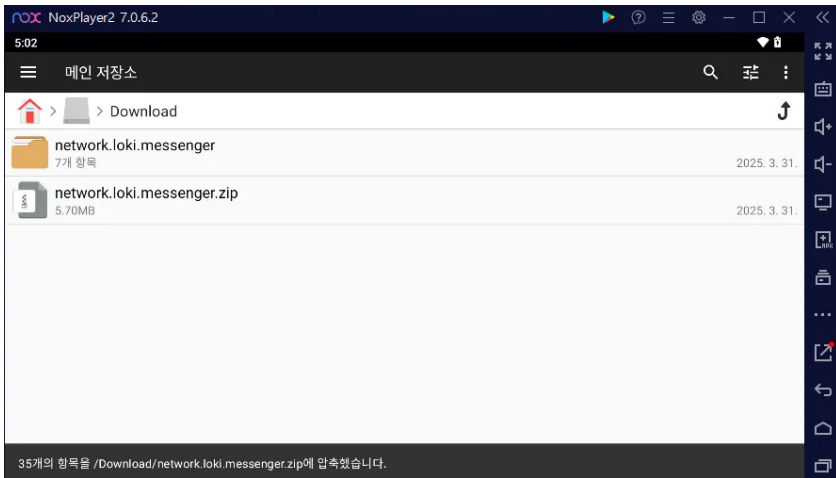
아티팩트 쌓기

Nox - 핸드폰끼리 일반 메시지, 사진, 음성 파일, 이모티콘 등 전송, 수신 후 삭제한 메시지 생성



아티팩트 추출

network.loki.messenger 폴더 Download로 복사, Windows로 이동



아티팩트 분석 (1)

app_parts: mms 파일 → 멀티미디어 파일(사진, 음성 파일), 암호화되어 있음

cache/log: HxD로 log 파일 암호화되어 있음을 확인

no_backup/android.work.workdb: 앱 설정과 관련된 내용. 대부분 비어 있음

The image displays a file system analysis of an Android application. On the left, a directory tree shows folders: `app_parts`, `cache`, `code_cache`, `databases`, `files`, `no_backup`, and `shared_prefs`. Red arrows point from these folders to a list of files on the right. The file list includes:

- `part1683593396130349772.mms`
- `part3063426406406665637.mms`
- `part5752846644491376763.mms`
- `part6408694575394946558.mms`
- `part8577096033215431479.mms`
- `androidx.work.workdb` (96KB)
- `com.google.android.gms.appid-no-backup` (0KB)

Below the file list, two screenshots of hex editors are shown. The left screenshot shows a hex editor with a search for `log-1743403113940` and a decoded text view. The right screenshot shows a hex editor with a search for `log-1743403113940` and a decoded text view. Red arrows connect the directory tree to the file list and the hex editors.

At the bottom, a string is shown:

```
<string name="pref_log_encrypted_secret">
{"data": "kfoqmISd4euvXm+RXD6PXL9ABHnkZb3IHtYBxellFaHuxkjqrnBsLe8G4hJPUfpTg", "iv": "8IF7R3T0x0+Ati7p"}</string>
```

아티팩트 분석 (2)

shared_prefs/network.loki.messenger_preferences.xml: 암호 정보, 민감한 정보 포함

pref_last_vacuum_time, pref_attachment_encrypted_secret, pref_profile_key,
pref_log_encrypted_secret, pref_database_encrypted_secret, pref_profile_name

databases/signal_v4.db: 데이터베이스가 암호화되어 있음

The diagram illustrates the file structure of an Android application and the analysis of two specific files. On the left, a file tree shows directories like app_parts, cache, code_cache, databases, files, no_backup, and shared_prefs. The databases directory contains signal_v4.db, signal_v4.db-shm, and signal_v4.db-wal. The shared_prefs directory contains several XML files, with network.loki.messenger_preferences.xml highlighted. Red arrows point from these files to a central image of a database viewer and a detailed XML snippet.

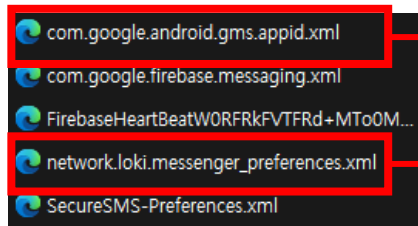
The central image shows a database viewer displaying the contents of signal_v4.db. The right side of the diagram shows the XML content of network.loki.messenger_preferences.xml, which is a preferences file for the Loki Messenger app. The XML contains various settings, including restoration_time, pref_is_using_fcm, pref_android_screen_lock, last_version_code, pref_last_vacuum_time, pref_incognito_keyboard, pref_trim_threads, pref_notification_channel_version, pref_enter_sends, pref_local_registration_id, pref_typing_indicators, pref_note_to_self_hidden, pref_read_receipts, pref_attachment_encrypted_secret, pref_autoplay_audio, pref_link_previews, pref_disable_passphrase, pref_profile_key, pref_profile_avatar_id, pref_log_encrypted_secret, pref_local_number, pref_database_encrypted_secret, and pref_profile_name. The pref_log_encrypted_secret and pref_database_encrypted_secret fields are highlighted with red boxes, indicating they contain sensitive information.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0"?>
<map>
  <long name="restoration_time" value="0"/>
  <boolean name="pref_is_using_fcm" value="true"/>
  <boolean name="pref_android_screen_lock" value="false"/>
  <int name="last_version_code" value="401"/>
  <long name="pref_last_vacuum_time" value="1743403117216"/>
  <boolean name="pref_incognito_keyboard" value="true"/>
  <boolean name="pref_trim_threads" value="false"/>
  <int name="pref_notification_channel_version" value="3"/>
  <boolean name="pref_enter_sends" value="false"/>
  <int name="pref_local_registration_id" value="1529"/>
  <boolean name="pref_typing_indicators" value="false"/>
  <boolean name="pref_note_to_self_hidden" value="true"/>
  <boolean name="pref_read_receipts" value="false"/>
  <string name="pref_attachment_encrypted_secret">
    {"data":"kdd356a15181CnRdnFRH1KjdWy/AcBb2F6AJVquunWJXyH33qDHT6F46CaJZNDL4ypT7Do2EbDPD1Q6KHkZNgEmrnk335a1uMFpiUJ5aT1S9MqaS5/gWwKePYSm3WRTL0VHrnyXwLX6X43S1I"}
  </string>
  <boolean name="pref_autoplay_audio" value="false"/>
  <boolean name="pref_link_previews" value="false"/>
  <boolean name="pref_disable_passphrase" value="true"/>
  <string name="pref_profile_key">dTKcjmwP077L8Zwz5SpLCnQ/gV//+ZSJ9UBYTetCwo</string>
  <int name="pref_profile_avatar_id" value="0"/>
  <boolean name="pref_migrated_to_group_v2_config" value="true"/>
  <string name="pref_log_encrypted_secret">{"data":"kfognISd4euvXm+RXD6PxlU3ABhJnkZb31Hy8XeWfaHuxkjqrnRsLe864hJPufpTg","iv":"81F7R3TQxQ+Ati7p"}</string>
  <boolean name="has_viewed_seed" value="true"/>
  <string name="pref_local_number">059075591817b4909a70f56e73da7fd10385e2459911981bd152549261d214d42b</string>
  <boolean name="pref_silent_notifications_enabled" value="false"/>
  <string name="pref_database_encrypted_secret">{"data":"ze7wmwGfvFx15KBUC/4gNQMS82R7SIowkJM9h2e5d2qUDZdRwsk1x8xy1JyRLTH/","iv":"D3DRepy11KsxAuH0"}</string>
  <string name="pref_profile_name">dhsama51</string>
  <long name="pref_last_version_check" value="1743403224819"/>
</map>
```

아티팩트 분석 (3)

shared_prefs/network.loki.messenger_preferences.xml의 pref_last_vacuum_time과
shared_prefs/ com.google.android.gms.appid.xml의 timestamp 비교
Vacuum time은 vacuum 작업(로컬 DB 최적화 관리) 시간

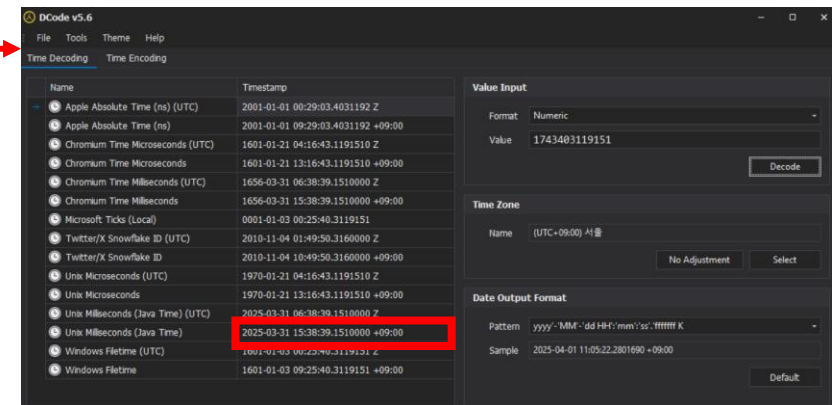
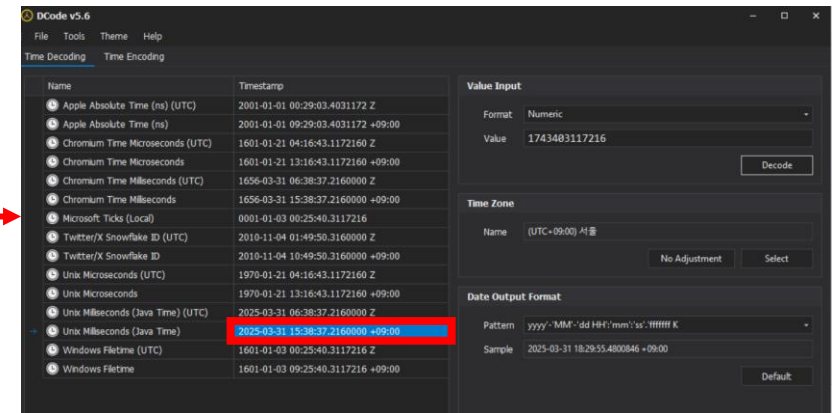


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<map>
  <string name="T|43512467490|+>"
    {"token":"cv0eFp0rTd6SRZRA80qCv5-APA91bHuDkCh2FxaH9u3a3aU5DE5FbP7z+LPB94qmSxTAI|usF3b8pttWV3U83AD12XGh34b9fZzhEXBucLPLvCe_3x-63U4XZJeZsJrxLXpDixvKcoV","appVersion":"4015","timestamp":1743403119151}
</map>
```

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<map>
  <long name="restoration_time" value="0"/>
  <boolean name="pref_is_using_fcm" value="true"/>
  <boolean name="pref_android_screen_lock" value="false"/>
  <int name="pref_last_version_check" value="1743403224819"/>
  <long name="pref_last_vacuum_time" value="1743403117216"/>
  <boolean name="pref_micromedia_keychain" value="true"/>
  <boolean name="pref_trim_threads" value="false"/>
  <int name="pref_notification_channel_version" value="3"/>
  <boolean name="pref_enter_sends" value="false"/>
  <int name="pref_local_registration_id" value="1529"/>
  <boolean name="pref_typing_indicators" value="false"/>
  <boolean name="pref_note_to_self_hidden" value="true"/>
  <boolean name="pref_read_receipts" value="false"/>
  <string name="pref_attachment_encrypted_secret">
    {"data":"kdd358a15181CnRdNFRH1KJdWY/AcB2F6JA/VquuUJyH33qQHT6F46CaJZNDL4ypT70c2E0BDP1Q9KHkZncgEmrk335a1uMfpiU5aT1S9Mqa35/gWkEpy3a3WRTLDVHmyXwLX6X43S1I"}
  </string>
  <boolean name="pref_autoplay_audio" value="false"/>
  <boolean name="pref_link_previews" value="false"/>
  <boolean name="pref_disable_passphrase" value="true"/>
  <string name="pref_profile_key">dTKcjwmp077L8Zvz5SpLcNq/gV//+ZSJ9UBVTetDwo=</string>
  <int name="pref_profile_avatar_id" value="0"/>
  <boolean name="migrated_to_group_v2_config" value="true"/>
  <string name="pref_log_encrypted_secret">{"data":"kfogmISd4euvXm+RXD6PxU3ABhJnkZb3IHyBXeWfHxukjqrnRSL664hJPUiPq","iv":"81F783TQxQ+Ati7p"}</string>
  <boolean name="has_viewed_seed" value="true"/>
  <string name="pref_local_number">059075591817b4903a70f56e73da7fd10385e2459911981bd152549261d214d42b</string>
  <boolean name="pref_call_notifications_enabled" value="false"/>
  <string name="pref_database_encrypted_secret">{"data":"ze7wmmGfVx15KBUc/4gNQM582R7S1owkI9M9h2e5d2qUDZ4Rwklx6xy1jyRLTH","iv":"D30Repy1IKsXAUH0"}</string>
  <string name="pref_profile_name">dhsana51</string>
  <long name="pref_last_version_check" value="1743403224819"/>
</map>
```



어플리케이션 분석 - 기초 조사 (1)

사전 정보

주요 데이터를 데이터베이스에 보관하고, 이를 암호화
AES256/GCM/NoPadding 사용

Android Keystore의 구조를 분석하여 암호키 추출

Keymaster: Android Keystore의 모든 키를 관리하는 키.

Blob 형태로 저장되어 있으며 다음과 같은 정보가 들어있음

- ①Android Keystore 버전 ②Nonce
- ③암호키 ④암호키 인증을 위한 Tag 및 속성 정보

→ Keymaster Blob이라고 불림

보안 인스턴트 메신저 Session 데이터베이스
복호화 방안 연구

박지수*, 박세준*, 김기윤**, 김종성***

*, **, ***국민대학교(학생, 대학원생, 교수)

Decryption Methods for Secure Instant Messenger Application
Session Database

JiSu Park*, SeJun Park*, GiYoon Kim**, JongSung Kim***

*, **, ***Kookmin University(Student, Graduate student, Professor)

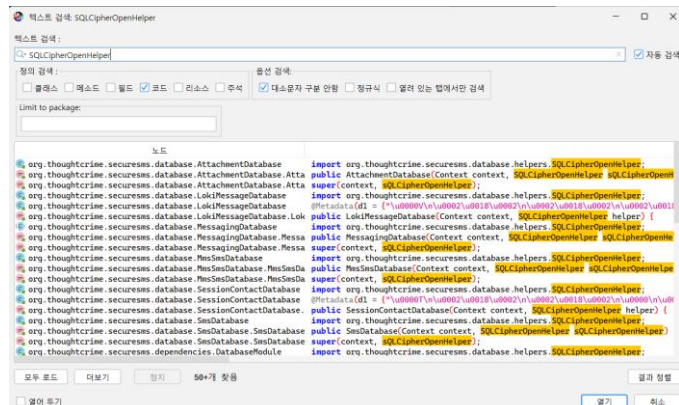
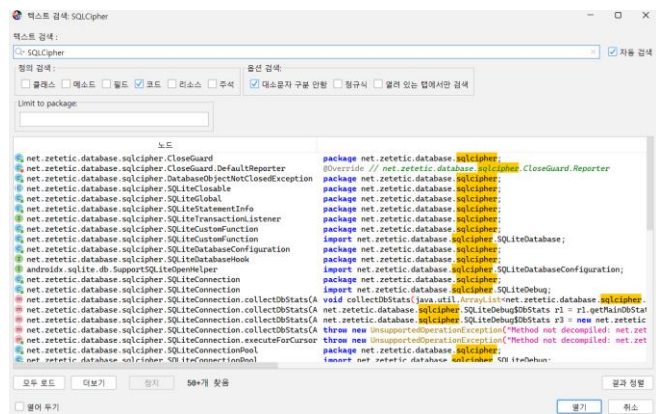
어플리케이션 분석 - 기초 조사 (2)

데이터베이스 암호화 정보 확인

SQLCipher로 암호화되어 있음 → 관련 코드 검색
암호화된 데이터베이스 이름인 signal_v4.db 확인

Jadx decompiler를 통해 분석해본 결과 signal_v4.db 데이터베이스는 SQLCipher로 암호화되었다. 다음은 실제 소스코드 상에서 확인 가능한 SQLCipher를 활용한 데이터베이스 오픈 과정이다(그림 2).

SQLCipher로 암호화되었다.



```
107 ▾ /* loaded from: classes4.dex */
public class SQLCipherOpenHelper extends SQLiteOpenHelper {
    private static final String CIPHER3_DATABASE_NAME = "signal.db";
    private static final String DATABASE_NAME = "signal_v4.db";
    private static final int DATABASE_VERSION = 69;
    private static final int MIN_DATABASE_VERSION = 28;
    private static final String TAG = "SQLCipherOpenHelper";
    private static final int lokiV10 = 31;
    private static final int lokiV11 = 32;
    private static final int lokiV12 = 33;
    private static final int lokiV13 = 34;
    private static final int lokiV14_BACKUP_FILES = 35;
    private static final int lokiV15 = 36;
    private static final int lokiV16 = 37;
    private static final int lokiV17 = 38;
    private static final int lokiV18_CLEAR_BG_POLL_JOBS = 39;
    private static final int lokiV19 = 40;
    private static final int lokiV20 = 41;
    private static final int lokiV21 = 42;
    private static final int lokiV22 = 43;
    private static final int lokiV23 = 44;
    private static final int lokiV24 = 45;
    private static final int lokiV25 = 46;
```

코드 Smali Simple Fallback ☐ Split view

어플리케이션 분석 - 기초 조사 (3)

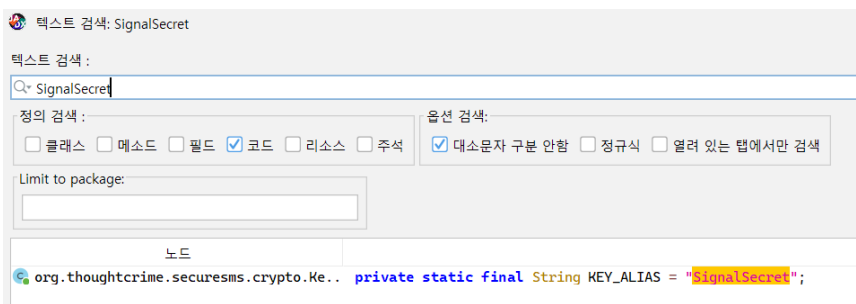
Session 암호키 정보

논문에 언급된 SignalSecret 검색 → 결과 1건

SignalSecret과 AndroidKeyStore이 단 1건 있고,

AES/GCM/NoPadding 사용을 직접 확인

SQLCipher에 사용된 패스프레이즈는 암호화되어 Shared Preference에 저장된다. 이때 암호 알고리즘은 AES/GCM/NoPadding 이었으며, 암호키는 Android Keystore에 저장되었다. Android Keystore에서 Session의 암호키를 구분하기 위한 별칭은 SignalSecret이다(그림 3).



```
/* loaded from: classes4.dex */
47 public final class KeyStoreHelper {
    private static final String ANDROID_KEY_STORE = "AndroidKeyStore";
    private static final String KEY_ALIAS = "SignalSecret";

48 public static SealedData seal(byte[] bArr) {
    SealedData sealedData;
49 SecretKey orCreateKeyStoreEntry = getOrCreateKeyStoreEntry();
    try {
        synchronized (CipherUtil.CIPHER_LOCK) {
56 Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
57 cipher.init(1, orCreateKeyStoreEntry);
62 sealedData = new SealedData(cipher.getIV(), cipher.doFinal(bArr));
        }
        return sealedData;
    } catch (InvalidKeyException | NoSuchAlgorithmException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e) {
65 throw new AssertionError(e);
    }
}
```

어플리케이션 분석 - 기초 조사 (4)

암호키를 안전하게 관리하는 시스템

키를 생성할 때 하드웨어 보안 모듈에 저장, 없다면 소프트웨어로 구현

KeyStore.getEntry(KEY_ALIAS, null)로 접근

→ KeyEntry 내부적으로 제공(Key를 뽑아낼 수는 없음)

cipher.init(1, keyStoreEntry, ...)로 keyEntry를 이용한 실제 암호화 진행,
cipher.init(2, keyStoreEntry, ...)로 keyEntry를 이용한 실제 복호화 진행

Nox에서는 /data/misc/keystore/ 경로에 persist.sqlite를 생성하여 키 저장, 관리
→ persist.sqlite와 관련된 코드는 앱 코드에 존재하지 않고, 직접 접근도 불가
내부적으로는 keyentry와 blobentry 테이블이 생성되어 있음



지원되는 알고리즘

- Cipher
- KeyGenerator
- KeyFactory
- KeyStore (KeyGenerator 및 KeyPairGenerator 와 동일한 키 유형 지원)
- KeyPairGenerator
- Mac
- Signature
- SecretKeyFactory

어플리케이션 분석 - Android Keystore에서 키 추출 (1)

Amaze 파일 매니저 이용해 persist.sqlite 추출

안드로이드 12 이상 버전에서의 암호키는
USERDATA/misc/keystore내 persist.sqlite 데
이터베이스에 저장된다. 해당 데이터베이스에서

고급

Use legacy listing for root
If enabled, uses legacy method to list files

루트 탐색기
루팅된 기기만 사용 가능합니다.

내부 공유 저장용량
125GB free of 126GB

최상위 폴더
465MB free of 2.11GB

sqlite

persistant.sqlite
4월 1일 | 오후 2:15 512B

공유폴더

로컬 파일을 애플레이어에 드래그 하시면 바로 안드로이드로 전송
이 가능합니다.

PC 폴더 오픈

안드로이드 폴더 오픈

.thumbnails

2025-04-01 오후 2:44

파일 폴더

persistant.sqlite

2025-04-01 오후 2:47

SQLITE 파일 108KB

어플리케이션 분석 - Android Keystore에서 키 추출 (2)

keyentry에서 SignalSecret 발견,
blobentry에서 같은 id를 가진 것 발견
0xD~0x1C에서 암호키 추출

→ '79 FA 7C 4A 90 A5 53 95 B1 1A 2B 49 7A 82 57 8D'

이름

타입

데이터베이스(T): keyentry

필터

id

key_type

domain

namespace

alias

데이터베이스(T): blobentry

필터

id

subcomponent_type

keyentryid

blob

데이터베이스(T): keyentry

필터

id

key_type

domain

namespace

alias

데이터베이스(T): blobentry

필터

id

subcomponent_type

keyentryid

blob

데이터베이스(T): keyentry

필터

id

key_type

domain

namespace

alias

데이터베이스(T): blobentry

필터

id

subcomponent_type

keyentryid

blob

데이터베이스(T): keyentry

필터

id

key_type

domain

namespace

alias

데이터베이스(T): blobentry

필터

id

subcomponent_type

keyentryid

blob

데이터베이스(T): keyentry

필터

id

key_type

domain

namespace

alias

데이터베이스(T): blobentry

필터

id

subcomponent_type

keyentryid

blob

Keymaster Blob을 획득하기 위해서는 keyentry 테이블과 blobentry 테이블을 활용해야 한다. 먼저, 두 테이블을 연결해주는 고유타가 필요하다. keyentry 테이블에는 alias 컬럼과 id 컬럼이 존재한다. alias 컬럼 내 Session 암호키의 별칭이 존재하는 열의 id 값이 두 테이블을 연결해주는 고유타가 된다. 해당 값을 blobentry 테이블의 keyentryid 컬럼에서 매칭한 뒤 같은 열에 존재하는 blob 데이터가 Keymaster Blob이 된다. 해당 Keymaster Blob의 offset 0xD~0x1C에는 암호키가 평문 형태로 저장되어있다(그림 5).

데이터베이스(T): keyentry

필터

id

key_type

domain

namespace

alias

state

im_uid

데이터베이스(T): blobentry

필터

id

subcomponent_type

keyentryid

blob

(그림 3) Android Keystore Key

어플리케이션 분석 - DB 암호화 흐름 (1)

1. signal_v4.db를 암호화할 DatabaseSecret 생성

```
private DatabaseSecret createAndStoreDatabaseSecret(Context context) {  
    byte[] bArr = new byte[32];  
    Util.SECURE_RANDOM.nextBytes(bArr);  
    DatabaseSecret databaseSecret = new DatabaseSecret(bArr);  
    TextSecurePreferences.setDatabaseEncryptedSecret(context, KeyStoreHelper.seal(databaseSecret.asBytes()).serialize());  
    return databaseSecret;  
}
```

SECRET_RANDOM으로 32byte(=256bit) 난수를 생성하고, DatabaseSecret에 저장

```
public class DatabaseSecret {  
    private final String encoded;  
    private final byte[] key;  
  
    public DatabaseSecret(byte[] bArr) {  
        this.key = bArr;  
        this.encoded = Hex.toStringCondensed(bArr);  
    }  
  
    public DatabaseSecret(String str) throws IOException {  
        this.key = Hex.fromStringCondensed(str);  
        this.encoded = str;  
    }  
  
    public String asString() {  
        return this.encoded;  
    }  
  
    public byte[] asBytes() {  
        return this.key;  
    }  
}
```

난수를 key와 encoded로 중복 저장

어플리케이션 분석 - DB 암호화 흐름 (2)

2. DatabaseSecret의 data를 AES/GCM/NoPadding 방식으로 암호화

```
private DatabaseSecret createAndStoreDatabaseSecret(Context context) {  
    byte[] bArr = new byte[32];  
    Util.SECURE_RANDOM.nextBytes(bArr);  
    DatabaseSecret databaseSecret = new DatabaseSecret(bArr);  
    TextSecurePreferences.setDatabaseEncryptedSecret(context, KeyStoreHelper.seal(databaseSecret.asBytes()).serialize(););  
    return databaseSecret;  
}
```

```
/* loaded from: classes4.dex */  
public final class KeyStoreHelper {  
    private static final String ANDROID_KEY_STORE = "AndroidKeyStore";  
    private static final String KEY_ALIAS = "SignalSecret";  
}
```

```
public static SealedData seal(byte[] bArr) {  
    SealedData sealedData;  
    SecretKey orCreateKeyStoreEntry = getOrCreateKeyStoreEntry();  
    try {  
        synchronized (CipherUtil.CIPHER_LOCK) {  
            Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");  
            cipher.init(1, orCreateKeyStoreEntry);  
            sealedData = new SealedData(cipher.getIV(), cipher.doFinal(bArr));  
        }  
        return sealedData;  
    } catch (InvalidKeyException | NoSuchAlgorithmException | BadPaddingException | IllegalBlockSizeException | NoSuchPaddingException e) {  
        throw new AssertionError(e);  
    }  
}
```

```
private static SecretKey getSecretKey(KeyStore keyStore) throws UnrecoverableKeyException {  
    try {  
        return ((KeyStore.SecretKeyEntry) keyStore.getEntry(KEY_ALIAS, null)).getSecretKey();  
    } catch (KeyStoreException e) {  
        e = e;  
        throw new AssertionError(e);  
    } catch (NoSuchAlgorithmException e2) {  
        e = e2;  
        throw new AssertionError(e);  
    } catch (UnrecoverableKeyException e3) {  
        throw e3;  
    } catch (UnrecoverableEntryException e4) {  
        e = e4;  
        throw new AssertionError(e);  
    }  
}
```

```
private static SecretKey getOrCreateKeyStoreEntry() {  
    return hasKeyStoreEntry() ? getKeyStoreEntry() : createKeyStoreEntry();  
}
```

```
private static SecretKey getKeyStoreEntry() {  
    KeyStore keyStore = getKeyStore();  
    try {  
        try {  
            return getSecretKey(keyStore);  
        } catch (UnrecoverableKeyException e) {  
            throw new AssertionError(e);  
        }  
    } catch (UnrecoverableKeyException unused) {  
        return getSecretKey(keyStore);  
    }  
}
```

```
private static SecretKey createKeyStoreEntry() {  
    try {  
        KeyGenerator keyGenerator = KeyGenerator.getInstance("AES", ANDROID_KEY_STORE);  
        keyGenerator.init(new KeyGenParameterSpec.Builder(KEY_ALIAS, 3).setBlockModes(CodePackage.GCM).setEncryptionPaddings("NoPadding").build());  
        return keyGenerator.generateKey();  
    } catch (InvalidAlgorithmParameterException | NoSuchAlgorithmException | NoSuchProviderException e) {  
        throw new AssertionError(e);  
    }  
}
```

DatabaseSecret을 암호화할 키(=KEY_ALIAS=SignalSecret)가 이미 있으면 가져오고, 없으면 Android Keystore에서 새로 생성함
Cipher라는 java 표준 라이브러리를 이용하며, Android Keystore에서 연산 수행

어플리케이션 분석 - DB 암호화 흐름 (3)

3. DatabaseSecret을 serialize함

```
private DatabaseSecret createAndStoreDatabaseSecret(Context context) {  
    byte[] bArr = new byte[32];  
    Util.SECURE_RANDOM.nextBytes(bArr);  
    DatabaseSecret databaseSecret = new DatabaseSecret(bArr);  
    TextSecurePreferences.setDatabaseEncryptedSecret(context, KeyStoreHelper.seal(databaseSecret.asBytes()).serialize());  
    return databaseSecret;  
}
```

iv와 암호화된 data를 serialize하여 JSON 형태로 변환

```
public String serialize() {  
    try {  
        return JsonUtil.toJsonThrows(this);  
    } catch (IOException e) {  
        throw new AssertionError(e);  
    }  
}
```

```
SealedData(byte[] bArr, byte[] bArr2) {  
    this.iv = bArr;  
    this.data = bArr2;  
}
```

SealedData를 serialize하여 반환

```
private static class ByteArraySerializer extends JsonSerializer<byte[]> {  
    private ByteArraySerializer() {}  
  
    @Override // com.fasterxml.jackson.databind.JsonSerializer  
    public void serialize(byte[] bArr, JsonGenerator jsonGenerator, SerializerProvider serializerProvider) throws IOException {  
        jsonGenerator.writeString(Base64.encodeToString(bArr, 3));  
    }  
}
```

str을 감지하면 호출되어 Base64 인코딩을 진행함

어플리케이션 분석 - DB 암호화 흐름 (4)

4. serialize 결과인 JSON 문자열을 SharedPreferences에 저장

```
private DatabaseSecret createAndStoreDatabaseSecret(Context context) {  
    byte[] bArr = new byte[32];  
    Util.SECURE_RANDOM.nextBytes(bArr);  
    DatabaseSecret databaseSecret = new DatabaseSecret(bArr);  
    TextSecurePreferences.setDatabaseEncryptedSecret(context, KeyStoreHelper.seal(databaseSecret.asBytes()).serialize());  
    return databaseSecret;  
}
```

```
public void setDatabaseEncryptedSecret(String secret) {  
    Intrinsics.checkNotNullParameter(secret, "secret");  
    setStringPreference("pref_database_encrypted_secret", secret);  
}
```

```
public final void setStringPreference(Context context, String key, String value) {  
    Intrinsics.checkNotNullParameter(context, "context");  
    PreferenceManager.getDefaultSharedPreferences(context).edit().putString(key, value).apply();  
}
```

매개변수 secret의 값을 pref_database_encrypted_secret에 저장

```
<string name="pref_database_encrypted_secret">  
{"data": "ze7wmwGfvFx15KBUC/4gNQm582R7S1owkJM9h2e5d2qUDZdRwxkIx8xyIjyRLTH/", "iv": "D3DRepyI1KsXAuH0"}</string>
```

어플리케이션 분석 - DB 복호화 흐름 (1)

1. SharedPreferences에 저장된 암호화된 패스프레이즈(JSON 문자열에 포함)를 로드

```
public DatabaseSecret getOrCreateDatabaseSecret() {
    String databaseUnencryptedSecret = TextSecurePreferences.getDatabaseUnencryptedSecret(this.context);
    String databaseEncryptedSecret = TextSecurePreferences.getDatabaseEncryptedSecret(this.context);
    if (databaseUnencryptedSecret != null) {
        return getUnencryptedDatabaseSecret(this.context, databaseUnencryptedSecret);
    }
    if (databaseEncryptedSecret != null) {
        return getEncryptedDatabaseSecret(databaseEncryptedSecret);
    }
    return createAndStoreDatabaseSecret(this.context);
}

private DatabaseSecret getUnencryptedDatabaseSecret(Context context, String str) {
    try {
        DatabaseSecret databaseSecret = new DatabaseSecret(str);
        TextSecurePreferences.setDatabaseEncryptedSecret(context, KeyStoreHelper.seal(databaseSecret.asBytes()).serialize());
        TextSecurePreferences.setDatabaseUnencryptedSecret(context, null);
        return databaseSecret;
    } catch (IOException e) {
        throw new AssertionError(e);
    }
}

private DatabaseSecret getEncryptedDatabaseSecret(String str) {
    return new DatabaseSecret(KeyStoreHelper.unseal(KeyStoreHelper.SealedData.fromString(str)));
}

private DatabaseSecret createAndStoreDatabaseSecret(Context context) {
    byte[] bArr = new byte[32];
    Util.SECURE_RANDOM.nextBytes(bArr);
    DatabaseSecret databaseSecret = new DatabaseSecret(bArr);
    TextSecurePreferences.setDatabaseEncryptedSecret(context, KeyStoreHelper.seal(databaseSecret.asBytes()).serialize());
    return databaseSecret;
}
```

Input : pref_database_encrypted_secret, Android_Keystore_Key
OutPut : Passphrase

- 1 : IV ← Base64_Decode(pref_database_encrypted_secret.iv)
- 2 : Data ← Base64_Decode(pref_database_encrypted_secret.data)
- 3 : Tag ← Data[-16:]
- 4 : Encrypted_key ← Data[0:48]
- 5 : Passphrase ← AES256/GCM/NoPadding_Decrypt
(Android_Keystore_Key, IV, Encrypted_key, Tag)
- 6 : return Passphrase

(그림 5) Getting PassPhrase of signal_v4.db

```
public String getDatabaseUnencryptedSecret() {
    return getStringPreference(pref_database_unencrypted_secret, null);
}
```

pref_database_unencrypted_secret이 없으므로 실패

```
public String getDatabaseEncryptedSecret() {
    return getStringPreference("pref_database_encrypted_secret", null);
}
```

```
public final String getStringPreference(Context context, String key, String defaultValue) {
    Intrinsic.checkNotNullParameter(context, "context");
    Intrinsic.checkNotNullParameter(key, "key");
    return PreferenceManager.getDefaultSharedPreferences(context).getString(key, defaultValue);
}
```

pref_database_encrypted_secret에서 JSON 문자열 로드

```
<string name="pref_database_encrypted_secret">
{"data":"ze7wmwGf vFx15KBUC/4gNQm582R7S1owkJM9h2e5d2qUDZdRwxk1x8xy1jyRLTH/","iv":"D3DRepy11KsXAuH0"}</string>
```

어플리케이션 분석 - DB 복호화 흐름 (2)

2. JSON 문자열(Base64 형태)을 SealedData(str 형태)로 deserialize

```
private DatabaseSecret getEncryptedDatabaseSecret(String str) {  
    return new DatabaseSecret(KeyStoreHelper.unseal(KeyStoreHelper.SealedData.fromString(str)));  
}
```

```
public static SealedData fromString(String str) {  
    try {  
        return (SealedData) JsonUtil.fromJson(str, SealedData.class);  
    } catch (IOException e) {  
        throw new AssertionError(e);  
    }  
}
```

JSON 문자열 → SealedData의 data, iv로 값을 넣어줌
data는 암호화된 패스프레이즈, iv는 data 암호화에 사용된 값

```
SealedData(byte[] bArr, byte[] bArr2) {  
    this.iv = bArr;  
    this.data = bArr2;  
}
```

```
private static class ByteArrayDeserializer extends JsonSerializer<byte[]> {  
    private ByteArrayDeserializer() {}  
  
    @Override // com.fasterxml.jackson.databind.JsonDeserializer  
    public byte[] deserialize(JsonParser jsonParser, DeserializationContext deserializationContext) throws IOException {  
        return Base64.decode(jsonParser.getValueAsString(), 3);  
    }  
}
```

byte[]임을 감지하면 호출되어 Base64 디코딩을 통해 str로 바뀌 줌

```
Input : pref_database_encrypted_secret, Android_Keystore_Key  
Output : Passphrase  
1 : IV ← Base64_Decode(pref_database_encrypted_secret.iv)  
2 : Data ← Base64_Decode(pref_database_encrypted_secret.data)  
3 : Tag ← Data[-16:]  
4 : Encrypted_key ← Data[0:48]  
5 : Passphrase ← AES256/GCM/NoPadding_Decrypt  
                    (Android_Keystore_Key, IV, Encrypted_key, Tag)  
6 : return Passphrase
```

어플리케이션 분석 - DB 복호화 흐름 (3)

3. SealedData 내 iv와 data를 AES/GCM/NoPadding 방식으로 복호화

```
private DatabaseSecret getEncryptedDatabaseSecret(String str) {  
    return new DatabaseSecret (KeyStoreHelper.unseal(KeyStoreHelper.SealedData.fromString(str)));  
}
```

```
public static byte[] unseal(SealedData sealedData) {  
    byte[] doFinal;  
    SecretKey keyStoreEntry = getKeyStoreEntry();  
    try {  
        synchronized (CipherUtil.CIPHER_LOCK) {  
            Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");  
            cipher.init(2, keyStoreEntry, new GCMParameterSpec(128, sealedData.iv));  
            doFinal = cipher.doFinal(sealedData.data);  
        }  
        return doFinal;  
    } catch (InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException |  
            throw new AssertionError(e);  
}
```

```
private static SecretKey getKeyStoreEntry() {  
    KeyStore keyStore = getKeyStore();  
    try {  
        try {  
            return getSecretKey(keyStore);  
        } catch (UnrecoverableKeyException e) {  
            throw new AssertionError(e);  
        }  
    } catch (UnrecoverableKeyException unused) {  
        return getSecretKey(keyStore);  
    }  
}
```

```
private static SecretKey getSecretKey(KeyStore keyStore) throws UnrecoverableKeyException {  
    try {  
        return ((KeyStore.SecretKeyEntry) keyStore.getEntry(KEY_ALIAS, null)).getSecretKey();  
    } catch (KeyStoreException e) {  
        e = e;  
        throw new AssertionError(e);  
    } catch (NoSuchAlgorithmException e2) {  
        e = e2;  
        throw new AssertionError(e);  
    } catch (UnrecoverableKeyException e3) {  
        throw e3;  
    } catch (UnrecoverableEntryException e4) {  
        e = e4;  
        throw new AssertionError(e);  
    }  
}
```

```
/* loaded from: classes4.dex */  
public final class KeyStoreHelper {  
    private static final String ANDROID_KEY_STORE = "AndroidKeyStore";  
    private static final String KEY_ALIAS = "SignalSecret";  
}
```

```
Input : pref_database_encrypted_secret, Android_KeyStore_Key  
Output : Passphrase  
1 : IV ← Base64_Decode(pref_database_encrypted_secret.iv)  
2 : Data ← Base64_Decode(pref_database_encrypted_secret.data)  
3 : Tag ← Data[-16:]  
4 : Encrypted_key ← Data[0:48]  
5 : Passphrase ← AES256/GCM/NoPadding_Decrypt  
                    (Android_KeyStore_Key, IV, Encrypted_key, Tag)  
6 : return Passphrase
```

SealedData를 복호화, new DatabaseSecret로 선언한 변수에 저장
getKeyStoreEntry()에서 복호화할 키(=KEY_ALIAS=SignalSecret)를 가져옴
Cipher라는 java 표준 라이브러리를 이용하며, Android Keystore에서 연산을 수행

어플리케이션 분석 - DB 생성, open, close, 조작 (1)

1. signal_v4.db를 생성해서 open하기

```
SQLiteDatabase beginTransaction() {  
    SQLiteDatabase writableDatabase = this.databaseHelper.getWritableDatabase()  
    writableDatabase.beginTransaction();  
    return writableDatabase;  
}
```

```
public SQLiteDatabase getWritableDatabase() {  
    SQLiteDatabase databaseLocked;  
    synchronized (this) {  
        databaseLocked = getDatabaseLocked(true);  
    }  
    return databaseLocked;  
}
```

```
private SQLiteDatabase getDatabaseLocked(boolean z) {  
    SQLiteDatabase sQLiteDatabase = this.mDatabase;  
    if (sQLiteDatabase != null) {  
        if (!sQLiteDatabase.isOpen()) {  
            this.mDatabase = null;  
        } else if (!z || !this.mDatabase.isReadOnly()) {  
            return this.mDatabase;  
        }  
    }  
    if (this.mIsInitializing) {  
        throw new IllegalStateException("getDatabase called recursively");  
    }  
    SQLiteDatabase sQLiteDatabase2 = this.mDatabase;  
    try {  
        this.mIsInitializing = true;  
        if (sQLiteDatabase2 != null) {  
            if (z && sQLiteDatabase2.isReadOnly()) {  
                sQLiteDatabase2.reopenReadWrite();  
            }  
        } else {  
            String str = this.mName;  
            if (str == null) {  
                sQLiteDatabase2 = SQLiteDatabase.create(null);  
            } else {  
                try {  
                    if (str.startsWith("/")) {  
                        str = this.mContext.getDatabasePath(str).getPath();  
                    }  
                    String str2 = str;  
                    File file = new File(str2).getParent();  
                    if (!file.exists()) {  
                        file.mkdirs();  
                    }  
                } catch (IOException e) {  
                    throw new RuntimeException(e);  
                }  
            }  
            sQLiteDatabase2 = SQLiteDatabase.openDatabase(str2, this.mPassword, this.mFactory, this.mEnableWriteAheadLogging ? 805306368 : 268435456, this.mErrorHandler, this.mDatabaseHook);  
        }  
    }  
    return sQLiteDatabase2;  
}
```

```
public static SQLiteDatabase openDatabase(String str, byte[] bArr, CursorFactory cursorFactory, int i, DatabaseErrorHandler databaseErrorHandler, SQLiteDatabaseHook sQLiteDatabaseHook)  
    SQLiteDatabase sQLiteDatabase = new SQLiteDatabase(str, bArr, i, cursorFactory, databaseErrorHandler, sQLiteDatabaseHook);  
    sQLiteDatabase.open();  
    return sQLiteDatabase;
```

```
public final SQLCipherOpenHelper provideOpenHelper(@ApplicationContext Context context) {  
    Intrinsics.checkNotNullParameter(context, "context");  
    DatabaseSecret orCreateDatabaseSecret = new DatabaseSecretProvider(context).getOrCreateDatabaseSecret();  
    SQLCipherOpenHelper migrateSqlCipher3To4IfNeeded(context, orCreateDatabaseSecret);  
    return new SQLCipherOpenHelper(context, orCreateDatabaseSecret);  
}
```

```
public SQLCipherOpenHelper(final Context context, DatabaseSecret databaseSecret) {  
    super(context, DATABASE_NAME, databaseSecret.asString(), (SQLiteDatabase.CursorFactory) null, 69, 28, (DatabaseErrorHandler) null);  
    @Override // net.zetetic.database.sqlcipher.SQLiteDatabaseHook  
    public void preKey(SQLiteConnection sQLiteConnection) {  
        SQLCipherOpenHelper.applySQLCipherPragmas(sQLiteConnection, true);  
    }  
    @Override // net.zetetic.database.sqlcipher.SQLiteDatabaseHook  
    public void postKey(SQLiteConnection sQLiteConnection) {  
        SQLCipherOpenHelper.applySQLCipherPragmas(sQLiteConnection, true);  
        if (System.currentTimeMillis() - TextSecurePreferences.getLastVacuumTime(context) > 604800000) {  
            sQLiteConnection.execute("VACUUM;", null, null);  
            TextSecurePreferences.setLastVacuumNow(context);  
        }  
    }  
    this.context = context.getApplicationContext();  
    this.databaseSecret = databaseSecret;  
}
```

```
public SQLiteOpenHelper(Context context, String str, byte[] bArr, SQLiteDatabase.CursorFactory cursorFactory, int i, DatabaseErrorHandler databaseErrorHandler, SQLiteDatabaseHook sQLiteDatabaseHook)  
    if (i < 1) {  
        throw new IllegalArgumentException("Version must be >= 1, was " + i);  
    }  
    this.mContext = context;  
    this.mName = str;  
    this.mPassword = bArr;  
    this.mFactory = cursorFactory;  
    this.mNewVersion = i;  
    this.mErrorHandler = databaseErrorHandler;  
    this.mDatabaseHook = sQLiteDatabaseHook;  
    this.mEnableWriteAheadLogging = z;  
    this.mMinimumSupportedVersion = Math.max(0, i2);  
}
```

어플리케이션 분석 - DB 생성, open, close, 조작 (2)

2. signal_v4.db close하기

```
void endTransaction(SQLiteDatabase sqLiteDatabase) {  
    sqLiteDatabase.setTransactionSuccessful();  
    sqLiteDatabase.endTransaction();  
}
```

```
public void endTransaction() {  
    acquireReference();  
    try {  
        getThreadSession().endTransaction(null);  
    } finally {  
        releaseReference();  
    }  
}
```

3. signal_v4.db sms 테이블 생성하기

```
public void onCreate(SQLiteDatabase sqLiteDatabase) {  
    sqLiteDatabase.execSQL(SmsDatabase.CREATE_TABLE);  
    sqLiteDatabase.execSQL(MmsDatabase.CREATE_TABLE);  
    sqLiteDatabase.execSQL(AttachmentDatabase.CREATE_TABLE);  
    sqLiteDatabase.execSQL(ThreadDatabase.CREATE_TABLE);  
    sqLiteDatabase.execSQL(DraftDatabase.CREATE_TABLE);  
    sqLiteDatabase.execSQL(PushDatabase.CREATE_TABLE);  
    sqLiteDatabase.execSQL(GroupDatabase.CREATE_TABLE);  
    sqLiteDatabase.execSQL(RecipientDatabase.CREATE_TABLE);  
    sqLiteDatabase.execSQL(GroupReceiptDatabase.CREATE_TABLE);  
    for (String str : SearchDatabase.CREATE_TABLE) {  
        sqLiteDatabase.execSQL(str);  
    }  
}
```

```
public class SmsDatabase extends MessagingDatabase {  
    ...
```

```
    public static final String CREATE_TABLE = "CREATE TABLE sms (_id integer PRIMARY KEY, thread_id INTEGER, address TEXT,  
    address_device_id INTEGER DEFAULT 1, person INTEGER, date INTEGER, date_sent INTEGER,  
    protocol INTEGER, read INTEGER DEFAULT 0, status INTEGER DEFAULT -1, type INTEGER,  
    reply_path_present INTEGER, delivery_receipt_count INTEGER DEFAULT 0, subject TEXT, body TEXT,  
    mismatched_identities TEXT DEFAULT NULL, service_center TEXT, subscription_id INTEGER DEFAULT -1,  
    expires_in INTEGER DEFAULT 0, expire_started INTEGER DEFAULT 0, notified DEFAULT 0,  
    read_receipt_count INTEGER DEFAULT 0, unidentified INTEGER DEFAULT 0,  
    is_deleted GENERATED ALWAYS AS ((type & 31) IN (28, 19)) VIRTUAL);";
```

어플리케이션 분석 - DB 생성, open, close, 조작 (3)

4. signal_v4.db sms 테이블에 insert

```
public long insertMessageOutbox(long j, OutgoingTextMessage outgoingTextMessage, boolean z, long j2, InsertListener insertListener, boolean z2) {
    long j3 = outgoingTextMessage.isSecureMessage() ? 10485782L : 22L;
    if (z) {
        j3 |= 64;
    }
    if (outgoingTextMessage.isOpenGroupInvitation()) {
        j3 |= 16384;
    }
    Address address = outgoingTextMessage.getRecipient().getAddress();
    Map<Address, Long> remove = earlyDeliveryReceiptCache.remove(j2);
    Map<Address, Long> remove2 = earlyReadReceiptCache.remove(j2);
    ContentValues contentValues = new ContentValues(6);
    contentValues.put("address", address.getAddress());
    contentValues.put("thread_id", Long.valueOf(j));
    contentValues.put("body", outgoingTextMessage.getMessageBody());
    contentValues.put("date", Long.valueOf(SnodeAPI.getNowWithOffset()));
    contentValues.put("date_sent", Long.valueOf(outgoingTextMessage.getSentTimestampMillis()));
    contentValues.put("read", (Integer) 1);
    contentValues.put("type", Long.valueOf(j3));
    contentValues.put(MmsSmsColumns.SUBSCRIPTION_ID, Integer.valueOf(outgoingTextMessage.getSubscriptionId()));
    contentValues.put("expires_in", Long.valueOf(outgoingTextMessage.getExpiresIn()));
    contentValues.put(MmsSmsColumns.EXPIRE_STARTED, Long.valueOf(outgoingTextMessage.getExpireStartedAt()));
    contentValues.put("delivery_receipt_count", Long.valueOf(Stream.of(remove.values()).mapToLong(new ToLongFunction() { // from class: org.thoughtcrime.se
        @Override // com.annimon.stream.function.ToLongFunction
        public final long applyAsLong(Object obj) {
            long longValue;
            longValue = ((Long) obj).longValue();
            return longValue;
        }
    }).sum()));
    contentValues.put("read_receipt_count", Long.valueOf(Stream.of(remove2.values()).mapToLong(new ToLongFunction() { // from class: org.thoughtcrime.secur
        @Override // com.annimon.stream.function.ToLongFunction
        public final long applyAsLong(Object obj) {
            long longValue;
            longValue = ((Long) obj).longValue();
            return longValue;
        }
    }).sum()));
    if (isDuplicate(outgoingTextMessage, j)) {
        Log.w(TAG, "Duplicate message (" + outgoingTextMessage.getSentTimestampMillis() + "), ignoring...");
        return -1L;
    }
    long insert = this.databaseHelper.getWritableDatabase().insert("sms", "address", contentValues);
    if (insertListener != null) {
        insertListener.onComplete();
    }
    if (z2) {
        DatabaseComponent.get(this.context).threadDatabase().update(j, true);
    }
    if (DatabaseComponent.get(this.context).threadDatabase().getLastSeenAndHasSent(j).first().longValue() < outgoingTextMessage.getSentTimestampMillis()) {
        DatabaseComponent.get(this.context).threadDatabase().setLastSeen(j, outgoingTextMessage.getSentTimestampMillis());
    }
    DatabaseComponent.get(this.context).threadDatabase().setHasSent(j, true);
    notifyConversationListeners(j);
    return insert;
}
```

어플리케이션 분석 - DB 생성, open, close, 조작 (4)

5. signal_v4.db sms 테이블에 update(읽음 표시)

```
public void incrementReceiptCount(MessagingDatabase.SyncMessageId syncMessageId, boolean z, boolean z2) {
    Cursor cursor;
    SQLiteDatabase writableDatabase = this.databaseHelper.getWritableDatabase();
    try {
        Cursor query = writableDatabase.query("sms", new String[]{"_id", "thread_id", "address", "type"}, "date_sent = ?", new String[]{String.valueOf(syncMessageId.getTimestamp())}, null, null, null);
        boolean z3 = false;
        while (query.moveToNext()) {
            try {
                if (MmsSmsColumns.Types.isOutgoingMessageType(query.getLong(query.getColumnIndexOrThrow("type")))) {
                    Address address = syncMessageId.getAddress();
                    Address fromSerialized = Address.fromSerialized(query.getString(query.getColumnIndexOrThrow("address")));
                    String str = z ? "delivery_receipt_count" : "read_receipt_count";
                    if (fromSerialized.equals(address)) {
                        long i = query.getLong(query.getColumnIndexOrThrow("thread_id"));
                        writableDatabase.execSQL("UPDATE sms SET " + str + " = " + str + " + 1 WHERE _id = ?", new String[]{String.valueOf(query.getLong(query.getColumnIndexOrThrow("_id")))});
                        DatabaseComponent.get(this.context).threadDatabase().update(j, false);
                        notifyConversationListeners(j);
                        z3 = true;
                    }
                }
            } catch (Throwable th) {
                th = th;
                cursor = query;
                if (cursor != null) {
                    cursor.close();
                }
                throw th;
            }
        }
        if (!z3) {
            if (z) {
                earlyDeliveryReceiptCache.increment(syncMessageId.getTimestamp(), syncMessageId.getAddress());
            }
            if (z2) {
                earlyReadReceiptCache.increment(syncMessageId.getTimestamp(), syncMessageId.getAddress());
            }
        }
        if (query != null) {
            query.close();
        }
    } catch (Throwable th2) {
        th = th2;
        cursor = null;
    }
}
```

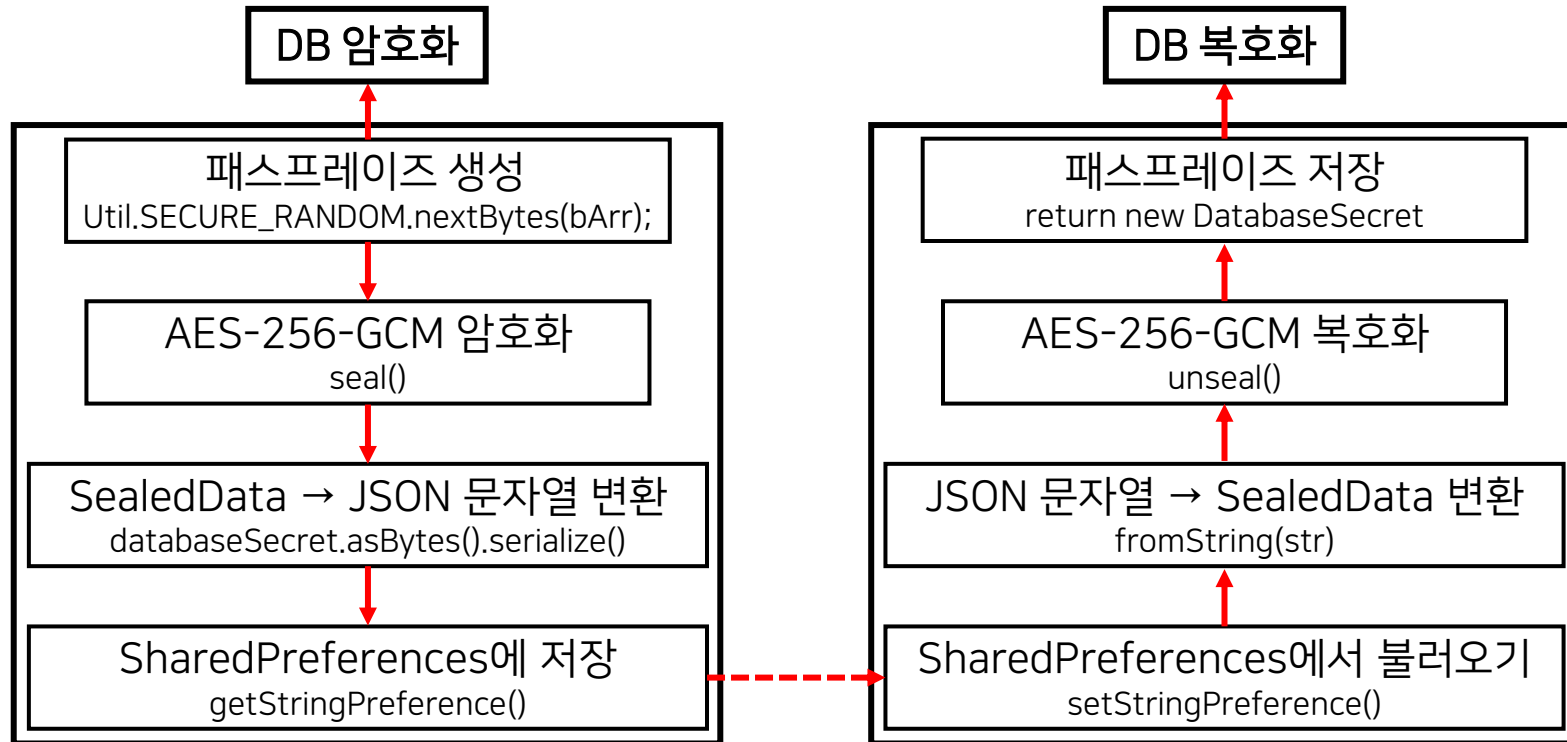
어플리케이션 분석 - DB 생성, open, close, 조작 (5)

6. signal_v4.db sms 테이블에 delete

```
public boolean deleteMessage(long j) {
    Log.i("MessageDatabase", "Deleting: " + j);
    SQLiteDatabase writableDatabase = this.databaseHelper.getWritableDatabase();
    long threadIdForMessage = getThreadIdForMessage(j);
    writableDatabase.delete("sms", "_id = ?", new String[]{j + ""});
    notifyConversationListeners(threadIdForMessage);
    return DatabaseComponent.get(this.context).threadDatabase().update(threadIdForMessage, false);
}
```

어플리케이션 분석 정리

Session 구조도



DB 복호화 (1)

패스프레이즈 획득

shared_prefs/network.loki.messenger_preferences.xml에서

pref_database_encrypted_secret에서 data, iv 획득

data: "ze7wmwGfvFx15KBUC/4gNQM582R7SlowkJM9h2e5d2qUDZdRwxklx8xyljyRLTH/"

암호화된 패스프레이즈 + GCM 태그 연접

iv: "D3DRepyl1KsXAuHO"

암호화에 사용된 iv가 Base64 인코딩되어 있음

```
<string name="pref_database_encrypted_secret">{"data":"ze7wmwGfvFx15KBUC/4gNQM582R7SlowkJM9h2e5d2qUDZdRwxklx8xyljyRLTH/","iv":"D3DRepyl1KsXAuHO"}</string>
```

```
<string name="pref_database_encrypted_secret">
```

```
{"data":"ze7wmwGfvFx15KBUC/4gNQM582R7SlowkJM9h2e5d2qUDZdRwxklx8xyljyRLTH/","iv":"D3DRepyl1KsXAuHO"}
```

```
</string>
```

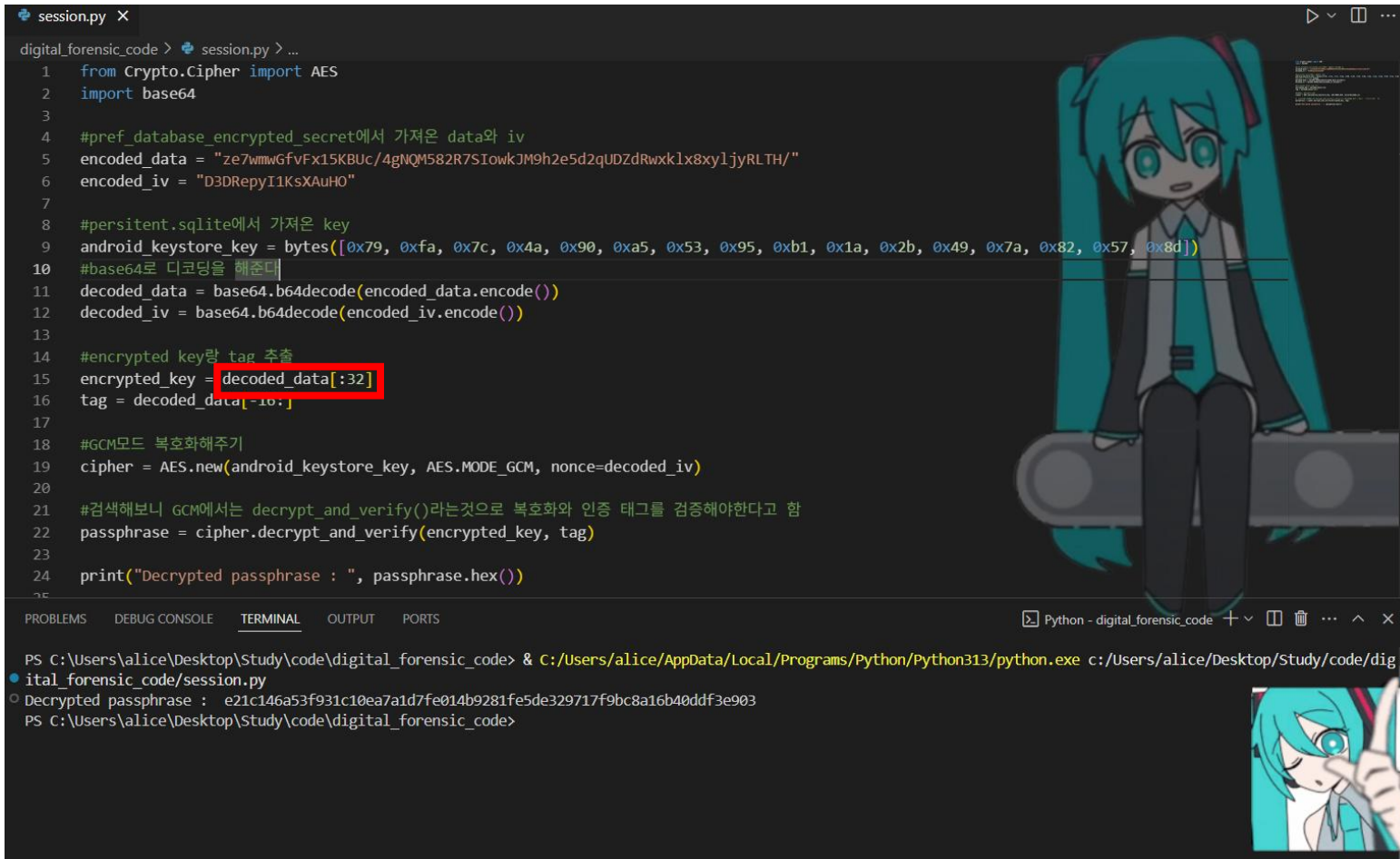
암호화된 패스프레이즈는 network.loki.messenger_preferences 내 pref_database_encrypted_secret 요소에 JSON 형태로 저장되어 있다(그림 6).

DB 복호화 (2)

패스프레이즈 복호화

Python 이용 - 결과: 'e21c146a53f931c10ea7a1d7fe014b9281fe5de329717f9bc8a16b40ddf3e903'

```
session.py X
digital_forensic_code > session.py > ...
1 from Crypto.Cipher import AES
2 import base64
3
4 #pref_database_encrypted_secret에서 가져온 data와 iv
5 encoded_data = "ze7mmwGfvFx15KBuc/4gNQm582R7SIowkJM9h2e5d2qUDZdRwxklx8xy1jyRLTH/"
6 encoded_iv = "D3DRepyI1KsXAuH0"
7
8 #persitent.sqlite에서 가져온 key
9 android_keystore_key = bytes([0x79, 0xfa, 0x7c, 0x4a, 0x90, 0xa5, 0x53, 0x95, 0xb1, 0x1a, 0x2b, 0x49, 0x7a, 0x82, 0x57, 0x8d])
10 #base64로 디코딩을 해준다
11 decoded_data = base64.b64decode(encoded_data.encode())
12 decoded_iv = base64.b64decode(encoded_iv.encode())
13
14 #encrypted key와 tag 추출
15 encrypted_key = decoded_data[:32]
16 tag = decoded_data[-16:]
17
18 #GCM모드 복호화해주기
19 cipher = AES.new(android_keystore_key, AES.MODE_GCM, nonce=decoded_iv)
20
21 #검색해보니 GCM에서는 decrypt_and_verify()라는것으로 복호화와 인증 태그를 검증해야한다고 함
22 passphrase = cipher.decrypt_and_verify(encrypted_key, tag)
23
24 print("Decrypted passphrase : ", passphrase.hex())
25
```



Input : pref_database_encrypted_secret, Android_Keystore_Key
OutPut : Passphrase
1 : IV \leftarrow Base64_Decode(pref_database_encrypted_secret.iv)
2 : Data \leftarrow Base64_Decode(pref_database_encrypted_secret.data)
3 : Tag \leftarrow Data[-16:]
4 : Encrypted_key \leftarrow Data[0:48]
5 : Passphrase \leftarrow AES256/GCM/NoPadding_Decrypt(Android_Keystore_Key, IV, Encrypted_key, Tag)
6 : return Passphrase

(그림 5) Getting PassPhrase of signal_v4.db

Data[0:48] \rightarrow 복호화된 패스프레이즈 길이 = 48

\neq

AES256/GCM/NoPadding 키 길이 = 32

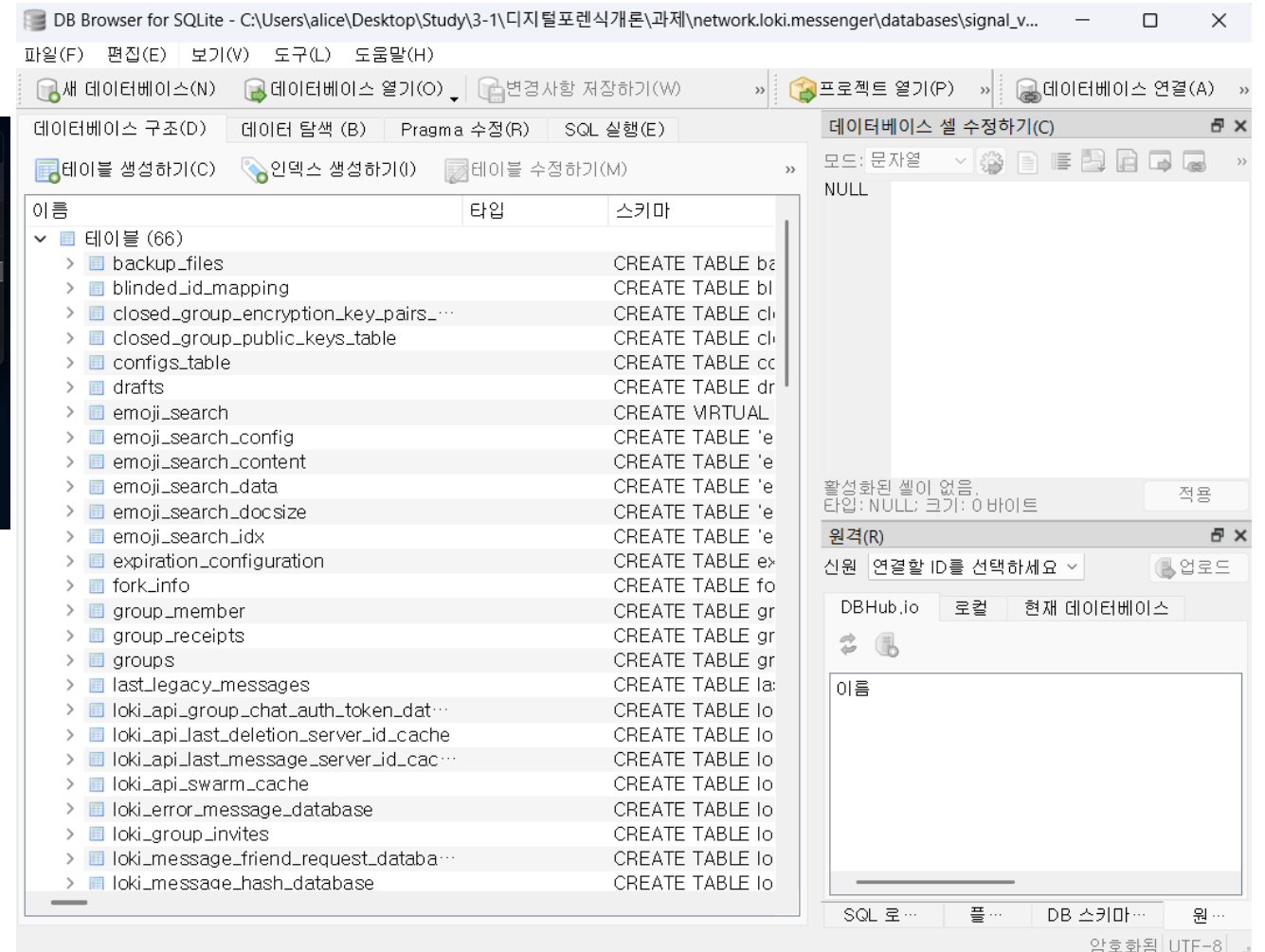
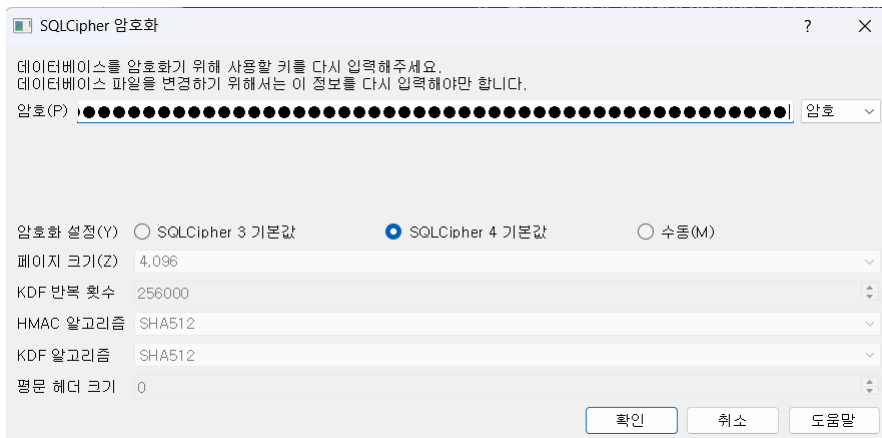
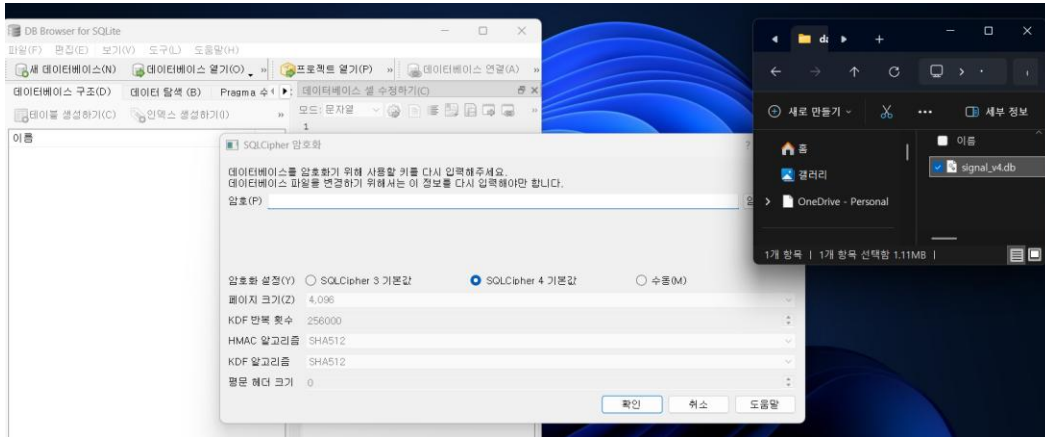
\rightarrow Data[0:32]로 진행

```
byte[] bArr = new byte[32];
Util.SECURE_RANDOM.nextBytes(bArr);
DatabaseSecret databaseSecret = new DatabaseSecret(bArr);
```

DB 복호화 (3)

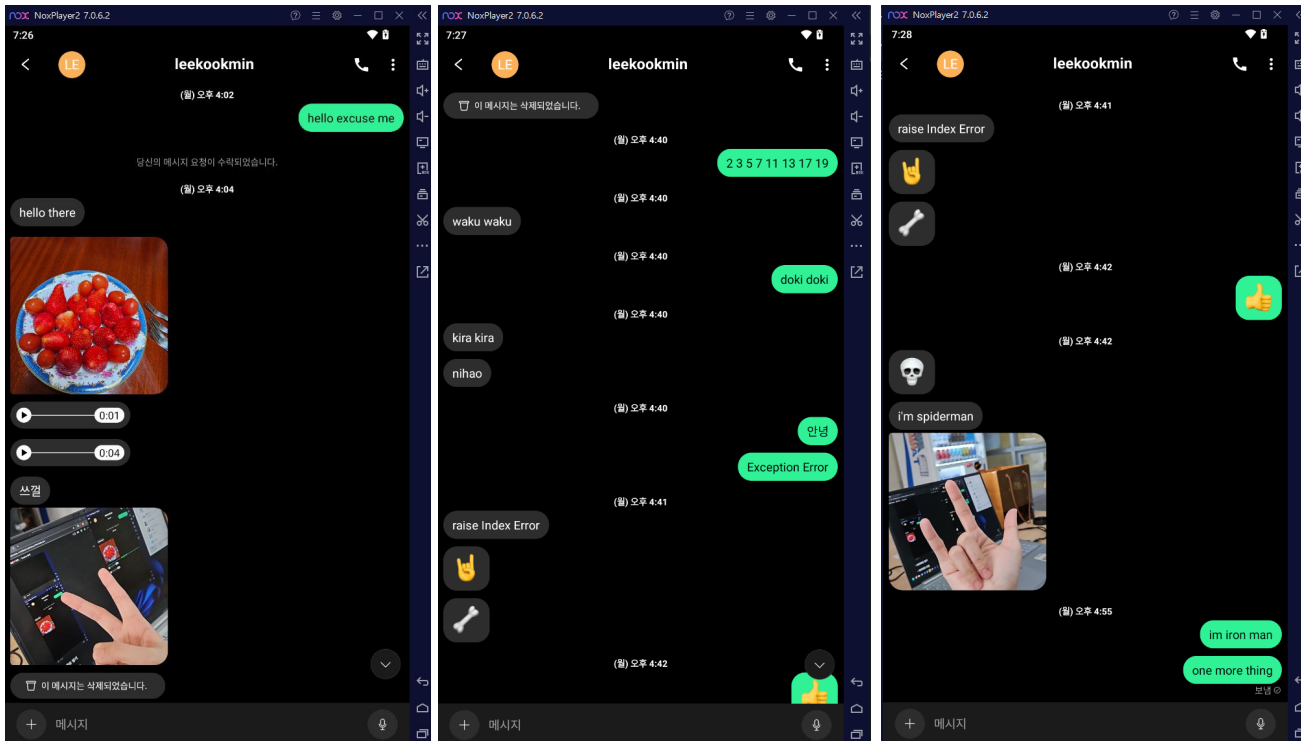
DB 복호화

패스프레이즈 입력을 통해 DB 복호화 확인



DB 복호화 (4)

복호화된 데이터베이스에서 대화 확인
주고받은 텍스트를 확인할 수 있었음



테이블(T): sms												
	sent	protocol	read	status	type	reply_path_present	delivery_receipt_count	subject	body	mismatched_identities	service_center	subscript
	필터	필터	필터	필터	필터	필터	필터	필터	필터	필터	필터	필터
1	9714414	NULL	1	-1	10485783	NULL	0	NULL	hello_world	NULL	NULL	
2	1522201	NULL	1	-1	10485783	NULL	0	NULL	hello excuse me	NULL	NULL	
3	1697984	31337	1	-1	10485780	1	0	NULL	hello there	NULL	GCM	
4	810215	NULL	1	-1	10485783	NULL	0	NULL	2 3 5 7 11 13 17 19	NULL	NULL	
5	818811	31337	1	-1	10485780	1	0	NULL	waku waku	NULL	GCM	
6	826267	NULL	1	-1	10485783	NULL	0	NULL	doki doki	NULL	NULL	
7	827020	31337	1	-1	10485780	1	0	NULL	kira kira	NULL	GCM	
8	832320	31337	1	-1	10485780	1	0	NULL	nihao	NULL	GCM	
9	852568	NULL	1	-1	10485783	NULL	0	NULL	안녕	NULL	NULL	
10	873138	NULL	1	-1	10485783	NULL	0	NULL	Exception Error	NULL	NULL	
11	878669	31337	1	-1	10485780	1	0	NULL	raise Index Error	NULL	GCM	
12	906442	31337	1	-1	10485780	1	0	NULL	👉	NULL	GCM	
13	921921	31337	1	-1	10485780	1	0	NULL	👉	NULL	GCM	
14	930209	NULL	1	-1	10485783	NULL	0	NULL	👉	NULL	NULL	
15	930513	31337	1	-1	10485780	1	0	NULL	👉	NULL	GCM	
16	716257	NULL	1	-1	10485783	NULL	0	NULL	im iron man	NULL	NULL	
17	951479	NULL	1	-1	10485783	NULL	0	NULL	one more thing	NULL	NULL	

테이블(T): mms									
	date	date_received	msg_box	read	m_jd	sub	sub_cs	body	
	필터	필터	필터	필터	필터	필터	필터	필터	
1	1743404684515	1743404684515	10551316	1	NULL	NULL	NULL	NULL	
2	1743404728201	1743404742385	10485780	1	NULL	NULL	NULL		
3	1743406194801	1743406200522	10485780	1	NULL	NULL	NULL		
4	1743406202226	1743406211511	10485780	1	NULL	NULL	NULL		
5	1743406345332	1743406355472	10485780	1	NULL	NULL	NULL	쓰겔	
6	1743406699338	1743406709698	10485779	1	NULL	NULL	NULL	이 메시지는 삭제되었습니다.	
7	1743407023074	1743407036407	10485780	1	NULL	NULL	NULL	i'm spiderman	

DB 내 sms 테이블 아티팩트 분석

Columns	Contents
_id	id
thread_id	1 st person / 2 nd person ...
address	person's address
address_device_id	1 (default, used in multi device)
person	0 (default, used in multi chat)
date	message save time
date_sent	message sent time
protocol	NULL(sent) / 31337(received)
read	0 (unread) / 1 (read)
status	-1 (default)
type	Received / Sent / Push / Encrypted
reply_path_present	NULL(sent) / 1(received)
delivery_receipt_count	'Delivery Receipt' count
subject	NULL (used in MMS as title)

body	message
mismatched_identities	NULL / 1(본인 키 불일치)
service_center	NULL(sent) / GCM(google cloud messaging)
subscription_id	-1 (default)
expires_in	0 / expiring time
expired_started	0 / expired time
notified	0 (default)
Read_receipt_count	'Read Receipt' count
unidentified	0 / 1(상대 공개키 불일치)
is_deleted	0 (default) / 1 (deleted)
reactions_unread	0 (unread) / 1 (read)
has_mention	0(default) / 1 (mentioned)
is_group_update	0 (default) / 1(updated)

Q&A

감사합니다

출처: 보안 인스턴트 메신저 Session에 대한 데이터베이스 복호화 방안 연구
<https://developer.android.com/privacy-and-security/keystore?hl=ko>