Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.3, Jun. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.3.369

Realm 데이터베이스 암·복호화 프로세스 및 기반 애플리케이션 분석*

윤 병 철,[1†] 박 명 서,[1] 김 종 성[2‡]
[1,2]국민대학교(대학원생, 교수)

Analysis of Encryption and Decryption Processes of Realm Database and Its Application*

Byungchul Youn,[1†] Myungseo Park,[1] Jongsung Kim[2‡]
[1,2]Kookmin University (Graduate student, Professor)

요 약

모바일 기기의 보편화로 스마트폰 보급률 및 사용률이 계속해서 증가하고 있으며, 애플리케이션에서 저장 및 관리해야 할 데이터 또한 증가하고 있다. 최근 애플리케이션은 효율적인 데이터 관리를 위해 모바일 데이터베이스를 이용하는 추세이다. 2014년 개발된 Realm 데이터베이스는 지속적인 업데이트와 빠른 속도, 적은 메모리 사용, 코드의 간결함과 가독성 등의 장점을 바탕으로 개발자들의 관심이 증가하고 있다. 또한, 데이터베이스에 저장된 개인정보의 기밀성과 무결성을 제공하기 위해 암호화를 지원한다. 하지만 암호화 기능은 안티 포렌식 기법으로 사용될 가능성이 있으므로 Realm 데이터베이스가 제공하는 암·복호화 동작 과정 분석이 필요하다. 본 논문에서는 Realm 데이터베이스의 구조와 암·복호화 동작 과정을 상세히 분석하였으며, 분석 내용에 관한 활용 사례를 보이기 위해 암호화를 지원하는 애플리케이션을 분석하였다.

ABSTRACT

Due to the widespread use of mobile devices, smartphone penetration and usage rate continue to increase and there is also an increasing amount of data that need to be stored and managed in applications. Therefore, recent applications use mobile databases to store and manage user data. Realm database, developed in 2014, is attracting more attention from developers because of advantages of continuous updating, high speed, low memory usage, simplicity and readability of the code. It also supports an encryption to provide confidentiality and integrity of personal information stored in the database. However, since the encryption can be used as an anti-forensic technique, it is necessary to analyze the encryption and decryption processes provided by Realm Database. In this paper, we analyze the structure of Realm Database and its encryption and decryption process in detail, and analyze an application that supports an encryption to propose the use cases of the Realm Database.

Keywords: Realm Database, Mobile Database, Digital Forensics

Received(01. 14. 2020), Modified(04. 06. 2020), Accepted(04. 20. 2020)

* 이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2017-0-00520, SCR-Friendly 대칭키 암호 및 응용모드 개발)

† 주저자, qjcjf123@kookmin.ac.kr
‡ 교신저자, jskim@kookmin.ac.kr(Corresponding author)

I. 서 론

개인 정보 유출 피해사례가 증가하면서 대부분의 애플리케이션은 이를 방지하기 위해 개인 정보를 암호화하여 디바이스나 서버에 저장한다. 개인 정보를 암호화하여 저장하게 되면 개인 정보가 타인에게 노출되더라도 평문 형태의 데이터를 획득할 수가 없다. 이러한 점은 포렌식 수사관에게 안티 포렌식으로 작용하며, 수사에 차질이 생기게 한다. 따라서 디지털 증거 획득을 위해서는 사용된 암호화 방식 분석 및 복호화 과정에 대한 연구가 선행되어야 한다. 하지만 애플리케이션 서비스마다 사용하는 데이터베이스와 암호화 방식은 서로 상이하며 이에 대한 연구사례도 적다. 따라서 본 논문은 모바일 애플리케이션에서 사용되는 Realm 데이터베이스의 구조를 분석과 암·복호화 방안에 대해 연구를 진행하였다.

Realm 데이터베이스는 C++ 기반의 오픈 소스 데이터베이스로써 모바일 기기에서 주로 사용되는 Android의 SQLite와 iOS의 Core Data를 대체하기 위한 목적으로 개발되었다[1]. 해당 데이터베이스는 제한된 배터리, 적은 메모리 자원 등과 같이 제한된 리소스 환경을 갖는 모바일 기기 내에서 빠른 속도를 제공하는 객체-지향 데이터베이스 (Object-Oriented Database)로써 데이터베이스 파일 내부의 모든 데이터를 객체 형태로 저장 및 관리한다. 또한, 5종류의 SDK (Software Development Kit)를 지원하여 특정 언어에 귀속되지 않고 Java, Swift, Object-C, C#, Javascript 등의 프로그램 언어를 통해 구현이 가능한 특징을 갖는다[2]. 또한, Android와 iOS 플랫폼을 대상으로 암호화를 지원하고 있으며, 지속적인 업데이트로 개발자들의 관심 또한 높아지고 있다[3].

기존 Realm 데이터베이스 관련 연구는 특정 플랫폼에서의 질의문 처리 속도 비교에 관한 연구로 편향되어 있으며, 최근 Realm 데이터베이스 구조와 삭제된 데이터 복구 방안에 관한 연구가 진행되었다.

Realm 데이터베이스의 전체 구조 분석과 삭제된 레코드를 복구하는 기법을 제시하였다[4]. 해당 논문의 경우 Realm 데이터베이스의 구조를 자세히 분석하였으나 Realm 데이터베이스의 객체 구조에 따른 데이터 접근 방식에 대한 언급이 부족하다. 따라서 본 논문에서는 기존 구조적 측면뿐 아니라 객체 구조별 데이터 접근 방식을 추가 서술하였다. Android 환경에서 ActiveAndroid, greenDAO, OrmLite, Sugar ORM, SQLite, Realm 데이터베이스를 대상으로 IRUD (Insert, Read, Update, Delete) 질의문 처리 시 발생하는 전력 소비량과 처리 시간을 측정 및 분석하였다[5]. Realm 데이터베이스 이외에도 SQLCipher의 암·복호화 동작 방식과 암호화된 데이터베이스 구조 분석을 통해 상용화 애플리케이션인 WeChat 메신저의 암호화된 데이터베이스 복호화에 성공한 사례도 존재한다[6].

본 논문은 총 5장으로 구성된다. 2장에서는 기존 Realm 데이터베이스 구조 분석 연구를 토대로 추가 사항을 제시하며, 3장에서는 Realm 데이터베이스의 암·복호화 동작 과정을 상세히 분석한다. 4장에서는 실제 사용되는 애플리케이션의 암호키 생성방식을 분석하고, 메모리 포렌식을 적용하여 암호키 획득 및 Realm 데이터베이스를 복호화한다. 그리고 마지막 5장에서는 결론을 맺는다.

II. Realm 데이터베이스 구조 분석

본 장에서는 기존의 Realm 데이터베이스 구조 분석 연구 결과와 함께 객체 구조별 데이터 접근 방식을 추가로 서술한다.

2.1 Realm 객체 구조

Realm 객체 구조는 데이터베이스 파일 전체의 구조를 의미하며 Fig. 1과 같이 Realm 헤더와 객체 컨테이너로 구성된다.

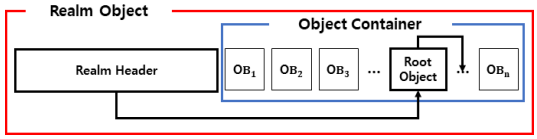

Fig. 1. Realm Object Structure

2.1.1 Realm 헤더

Realm 객체의 구성 요소인 Realm 헤더는 24바이트 크기를 갖는 메타데이터로써 오프셋, 시그니처, 파일 포맷, Flag 정보를 갖는다(Table 1).

해당 데이터의 0x00 위치에는 8바이트 크기의 루트 객체 오프셋이 2개 저장된다. 2개의 루트 객체를 갖는 이유는 Realm 데이터베이스가 데이터 쓰기(수정, 삽입, 삭제) 과정 중 COW[1] (Copy On Write) 기

Table 1. Structure of Realm Header

Offset	Size (Byte)	Field
0x00	8	#1 Root Object Offset
0x08	8	#2 Root Object Offset
0x10	4	Mnemonic(Signature)
0x14	2	File Format
0x16	1	Reserved Area
0x17	1	Flag

법을 사용하기 때문이다. 따라서 데이터로 접근하기 위해서는 0x17 위치의 Flag를 통해서 참조할 루트 객체를 결정해야 한다. Flag 값은 '0' 혹은 '1'의 값을 가지며 '0'일 경우 #1 루트 객체를 참조하고, '1'일 경우 #2 루트 객체를 참조한다. 또한, 0x10 위치에서 시그니처인 'T-DB'를 확인할 수 있다[4].

2.2 객체 구조

본 절에서의 객체는 객체 컨테이너에 존재하는 모든 객체를 의미하며, 기본적인 객체의 구조는 Fig. 2와 같다.

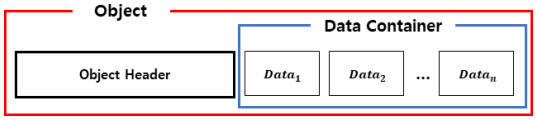

Fig. 2. Object Structure

2.2.1 객체 헤더

모든 객체는 동일 구조의 헤더를 갖는다. 내부 구조는 Table 2와 같으며, 첫 4바이트는 체크섬을 의미한다. 해당 값은 데이터 컨테이너의 체크섬을 나타내지만, 아직 구현되지 않아 시그니처 형태인 'AAAA'가 저장된다[7]. 체크섬 이후에는 해당 객체의 메타 정보가 저장된 Flag와 데이터 컨테이너의 크기 혹은 내부 원소의 개수를 나타내는 Size가 저장된다.

Table 2에서 언급한 Flag는 객체의 메타 정보를 갖는 1바이트 크기의 데이터로써, 비트 단위로 총 5가지의 정보를 갖는다(Table 3).

1) 데이터 쓰기 작업 시, 원본 데이터를 대상으로 쓰기 동작을 진행하지 않고, 자신의 복사본을 생성한 후 복사본에 쓰기 행위를 진행하는 방법

Table 2. Structure of Object Header

Offset	Size (Byte)	Field	Content
0x00	4	Checksum	Not yet Implemented
0x04	1	Flag	Object Information
0x05	3	Size	Use to calculate size of Data Container

Table 3. Structure of Flag

Num	Size(Bit)	Field
1	1	is_inner_bptree_node
2	1	has_refs
3	1	context_flag
4	2	width_scheme
5	3	width_ndx

특히 width_ndx와 width_scheme 값을 이용한다면 객체 헤더 이후 데이터 컨테이너의 전체크기 및 내부 데이터의 폭을 계산할 수 있어 원하는 데이터로의 접근이 가능하다. 예를 들어 Flag의 값이 '0100 0101'이라고 가정한다면, 해당 객체의 width_ndx와 width_scheme의 값은 각각 '5', '0'임을 알 수 있다. 또한, Table 4와 Table 5를 통해서 데이터 컨테이너 내부 데이터의 길이가 2바이트이며, 객체 헤더 이후 데이터 컨테이너의 전체크기는 2×Size 바이트임을 확인할 수 있다.

Table 4. width_ndx

width_ndx	0	1	2	3	4	5	6	7
Value of 'width'	0	1	2	4	8	16	32	64

Table 5. width_scheme

Value	Meaning of 'width'	Number of Bytes used after header
0	Number of Bits	ceil[2](width×Size/8)
1	Number of Bytes	width×Size
2	Ignored	Size

2) C++/C 언어에서의 올림 함수

III. Realm 데이터베이스 암·복호화 동작 과정

본 장에서는 Realm-Core 오픈 소스 분석을 통해 Realm 데이터베이스의 암호화 동작 과정을 상세히 분석하였다. 또한, 암호화된 데이터베이스의 구조를 파악하고 이를 통해 복호화 과정을 밝혔다[7]. 상세 내용은 다음과 같다.

3.1 암호화 동작 과정

Realm 데이터베이스의 암호화 동작 과정은 Fig. 3과 같다. Realm 데이터베이스를 암호화하기 위해서는 64바이트의 암호키가 필요하다. 64바이트 암호키는 두 가지 용도의 키를 포함하고 있으며, 상위 32바이트는 AES256-CBC 키, 하위 32바이트는 HMAC-SHA224 (이후 HMAC으로 명칭한다.) 키이다.

3.1.1 IV_Table

암호화 과정 진행 전 Realm 데이터베이스는 IV_Table 구조체를 호출한다. IV_Table의 내부 구성 요소는 Table 6과 같으며 32바이트를 기준으로 현재와 과거의 정보를 저장한다. 상위 32바이트에는 현재 암호화에 관련된 정보가 암호화 동작 시 업데이트되어 저장된다. 그 중 iv1은 암호화에 사용되는 IV (Initial Vector)의 상위 4바이트로 사용되며, 이후 28바이트의 hmac1에는 암호화된 블록의 HMAC 값이 업데이트된다. 하위 32바이트에는 iv1과 hmac1의 업데이트 이전의 값이 복사되어 저장된다. 최초 암호화 과정에서는 IV_Table의 모든 필드 값은 0으로 초기화되며, 암호화 대상 블록마다 서로 다른 IV_Table을 호출한다.

Input : *Realm Database*, *Encryption Key*
Output : *Encrypted Realm Database*
Variable : *Realm Database Block* $= \{R_0, R_1, \dots, R_{n-1}\}$
Encrypted Realm Database Block $= \{C_0, C_1, \dots, C_{n-1}\}$
IV_Table $= iv1 \,\|\, hmac1 \,\|\, iv2 \,\|\, hmac2$

1: $i \leftarrow 0$
2: $Pos \leftarrow 0$
3: *AES Key* $\leftarrow$ *Upper* 32 *bytes of Encryption Key*
4: *HMAC Key* $\leftarrow$ *Lower* 32 *bytes of Encryption Key*
5: *Intialize IV_Table as* 0
6: *While*($i < n$) :
7: *Get IV_Table*
8: *Copy IV_Table.iv*2 $\leftarrow$ *IV_Table.iv*1
9: *Copy IV_Table.hmac*2 $\leftarrow$ *IV_Table.hmac*1
10: *Do* :
11: *Update IV_Table.iv*1 $\leftarrow$ *IV_Table.iv*1 + 1
12: *Compute IV* $\leftarrow$ *IV_Table.iv*1 $\|$ *Pos* $\|$ 0 0 ... 0
13: *Encrypt* $C_i \leftarrow AES256-CBC(R_i, AES\ Key, IV)$
14: *Update IV_Table.hmac*1 $\leftarrow HMAC-SHA224(C_i, HMAC\ Key)$
15: *While* (*Upper* 4 *bytes of IV_Table.hmac*1 and *IV_Table.hmac*2 *is equal*)
16: *Write IV_Table* and C_i *in Encrypted Realm Database*
17: $Pos \leftarrow Pos + 4096$
18: $i \leftarrow i + 1$
19: *end While*
20: *Return Encrypted Realm Database*

Fig. 3. Encryption Process of Realm Database

Table 6. Structure of IV_Table

Offset	Size (Byte)	Field	Content
0x00	4	iv1	The first 4 bytes of IV using current encryption
0x04	28	hmac1	HMAC value of currently encrypted block
0x32	4	iv2	The first 4 bytes of IV using previous encryption
0x36	28	hmac2	HMAC value of previous encrypted block

3.1.2 AES256-CBC 암호화

IV_Table 구조체를 호출한 후, Realm 데이터베이스는 iv1과 hmac1을 iv2와 hmac2 위치에 복사한다. 이후, iv1 값을 1만큼 증가시키며, 입력받은 암호키의 상위 32바이트를 고정키로 사용하여 모든 블록을 암호화한다. 사용되는 알고리즘은 AES256-CBC/Zero-Padding 암호화 알고리즘을 사용하며, 4,096바이트 크기의 데이터가 한 블록으로 기본 설정되어 있다. 또한, 암호화에 사용되는 IV는 블록마다 새롭게 생성한다. 첫 4바이트는 IV_Table의 iv1이 위치하며, 이후 4바이트는 해당 블록의 위치인 Pos, 하위 8바이트는 0으로 초기화한다. 블록의 위치를 나타내는 Pos는 시작 블록을 기준으로 0으로 초기 설정되어 있으며, 이후 블록에 대해서는 4,096 크기만큼 증가시킨다. 즉 N 번째 블록을 암호화하는데 필요한 Pos 값은 (N-1)×4,096이다. Table 7은 IV의 구조를 나타낸다.

Table 7. Structure of IV

Offset	Size (Byte)	Content
0x00	4	iv1
0x04	4	Pos=(N-1)×4,096 N: Block number
0x08	8	0

3.1.3 HMAC-SHA224 계산

블록 암호화 과정이 종료되면, Realm 데이터베이스는 초기 입력받은 암호키의 하위 32바이트를 HMAC 키로 이용하여 암호화된 Realm 데이터베이스 블록인 C_i의 HMAC 값을 계산한다. 이후 계산된 HMAC 값을 IV_Table의 hmac1 위치에 업데이트하며 해당 결과값과 hmac2 값이 동일한지 확인한다. 만약 두 값의 상위 4바이트가 같다면 iv1을 1 증가시켜 다시 암호화를 진행하며, 다를 시에는 다음 블록에 대해서 암호화를 진행한다.

3.1.4 암호화된 Realm 데이터베이스 구조 분석

암호화된 Realm 데이터베이스의 구조는 IV_Table 컨테이너와 암호화된 블록 컨테이너로 나누어진다. 블록당 암호화 동작 과정이 종료되면 Realm 데이터베이스는 업데이트된 IV_Table과 암호화된 블록 순서로 쓰기 동작을 수행한다. 암호화된 Realm 데이터베이스의 구조는 Fig. 4와 같이 0x00 위치부터 4,096바이트 크기만큼 IV_Table을 차례로 저장하며, 이후 0x1000 위치부터 IV_Table 순서에 매칭되는 암호화된 블록을 저장한다.

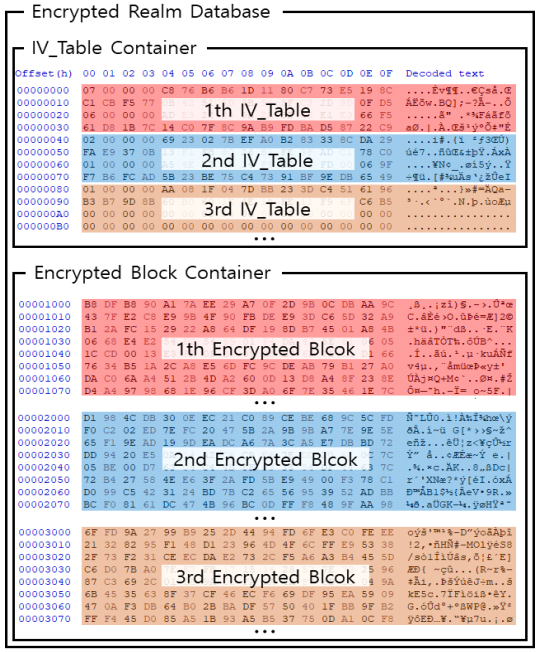

Fig. 4. Encrypted Realm Database Structure

이때 IV_Table 컨테이너와 암호화된 블록 컨테이너에 저장할 수 있는 IV_Table과 암호화된 블록의 최대 개수는 64개이다. 이를 초과할 시에는 암호화된 블록의 끝 주소부터 같은 형식으로 IV_Table과 암호화된 블록이 저장된다.

3.2 복호화 동작 과정

Realm 데이터베이스 복호화를 수행하기 위해서는 암호화에 사용된 64바이트 암호키가 필요하며, 복호화 동작 과정은 Fig. 5와 같다. 암호화된 Realm 데이터베이스 파일과 암호키를 입력받으면 우선 암호화된 Realm 데이터베이스에 저장된 IV_Table 구조체를 호출하여 iv1 값이 '0'임을 확인한다. 만약 '0'이라면 암호화가 일어나지 않았다고 판단하여 복호화를 중지한다. 반대로 '0'이 아니라면 입력받은 암호키의 하위 32바이트를 사용하여 암호화된 블록의 HMAC 값(이하 hmac이라 명칭 한다.)을 계

Input : *Encrypted Realm Database*, *Encryption Key*
Output : *Decrypted Realm Database*
Variable : *Decrypted Realm Database Block* = $\{D_0, D_1, \ldots, D_{n-1}\}$
Encrypted Realm Database Block = $\{C_0, C_1, \ldots, C_{n-1}\}$
IV_Table = $iv1 \| hmac1 \| iv2 \| hmac2$

```
1 : i ← 0
2 : Pos ← 0
3 : AES Key ← Upper 32 bytes of Encryption Key
4 : HMAC Key ← Lower 32 bytes of Encryption Key
5 : While( i < n) :
6 :     Get IV_Table from Decrypted Realm Database
7 :     if IV_Table.iv1 is 0 :
8 :         Return False
9 :     end if
10 :    Compute hmac ← HMAC−SHA224(C_i, HMAC Key)
11 :    if hmac and IV_Table.hmac1 is not equal
12 :        if IV_Table.iv2 is 0 :
13 :            Return False
14 :        end if
15 :        if hmac and IV_Table.hmac2 is equal :
16 :            IV_Table.iv1 ← IV_Table.iv2
17 :            IV_Table.hmac1 ← IV_Table.hmac2
18 :        else :
19 :            Return False
20 :        end if
21 :    end if
22 :    Compute IV ← IV_Table.iv1 ‖ Pos ‖ 0 0 ... 0
23 :    Decrypt D_i ← AES256−CBC(C_i, AES Key, IV)
24 :    Write D_i in Decrypted Realm Database
25 :    Pos ← Pos + 4096
26 :    i ← i + 1
27 : end While
28 : Return Decrypted Realm Database
```

Fig. 5. Decryption Process of Realm Database

Table 8. Structure of Encrypted Realm Database Encryption Key

	Column Name	Content
AES Key	item_key	$HexString(AES\,256-CBC(\text{“}aes\text{”}))$
	item_value	$IV \parallel AES\,256-CBC(\ AES\ Key \parallel HexString(SHA\,256(\ AES\ Key)))$
HMAC Key	item_key	$HexString(AES\,256-CBC(\text{“}hmac\text{”}))$
	item_value	$IV \parallel AES\,256-CBC(\ HMAC\ Key \parallel HexString(SHA\,256(\ HMAC\ Key)))$

산하며, hmac1과 비교한다. 만약 두 값이 같다면 데이터의 위·변조가 일어나지 않았으며, 암호화된 블록을 쓰는 과정 중 오류가 발생하지 않았다고 판단하여 정상적으로 복호화를 진행한다. 복호화 시에는 입력받은 암호키의 상위 32바이트를 AES 키로 사용하며, iv1을 통해 IV를 재생성한 후, 복호화를 진행한다.

만약 두 값이 다르다면 hmac과 hmac2를 비교한다. 이는 암호화 과정 중 업데이트된 IV_Table을 저장한 후, 암호화된 블록을 저장하는 과정 중 발생할 수 있는 오류에 대응하기 위한 Fault Tolerance[3] 기능으로 iv1을 통해서 복호화를 진행할 수 없더라도 iv2를 사용해 정상적으로 복호화가 진행될 수 있게 한다. 만약 두 값이 다르다면 복호화 과정을 종료한다.

IV. Realm 데이터베이스 기반 애플리케이션 분석

본 장에서는 앞선 장에서 언급한 Realm 데이터베이스 암·복호화 동작 과정을 토대로 SAP Concur 애플리케이션의 암호키 생성 과정을 분석하였다. 또한, 메모리 포렌식을 통해 사용된 암호키 재현과 복호화에 성공하였다. 상세 내용은 다음과 같다.

4.1 암호화된 데이터 식별

SAP Concur는 500만 이상의 다운로드 수를 기록한 출장 및 경비 관리 자동화 솔루션 애플리케이션이며, 암호화된 Realm 데이터베이스를 통해 사용자의 정보를 저장한다. 해당 애플리케이션에 로그인하게 되면 /data/data/com.concur.breeze/files/encrypted-realm/ 경로에 storage-expense-db.realm 파일이 자동 생성되며, 사용자가 제출한 경비 지출 내용, 경비 종류 등의 개인 정보가 암호화되어 저장된다.

3) 일부 결함 또는 고장이 발생하여도 정상적 혹은 부분적으로 기능을 수행할 수 있게 하는 기능

4.2 암호키 생성 과정 분석

SAP Concur 애플리케이션은 Realm 데이터베이스 암호화에 사용되는 AES 키와 HMAC 키를 JAVA의 SecureRandom을 통해 각각 생성한다. 생성된 두 키는 SHA256 알고리즘에 의해 해싱된 결과값과 함께 연접되어 Keystore에 존재하는 암호키와 AES256-CBC/PKCS#5-Padding 알고리즘으로 암호화된 후 암호화에 사용된 IV와 연접되어 저장된다. 또한, 암호화된 AES 키와 HMAC 키를 식별하기 위해서 “aes”와 “hmac” 문자열을 동일한 방식으로 암호화하며, 저장 시 애플리케이션 패키지 내에 존재하는 DataVault 데이터베이스 파일 내부 DATA_VAULT_2 테이블의 item_key와 item_value 칼럼에 암호화된 문자열과 키값을 각각 저장한다(Table 8). Realm 데이터베이스에 쓰기 동작을 수행하는 경우 Keystore를 이용하여 item_key와 item_value 칼럼의 암호화된 데이터를 복호화를 진행하여 AES 키와 HMAC 키를 획득한 후 데이터베이스를 복호화한다. 해당 과정에 대한 상세 내용은 본 논문의 연구 범위를 벗어나므로 생략한다.

4.3 암호키 재현 및 복호화

본 절에서는 메모리 포렌식을 통한 암호키 재현과 암호화된 데이터베이스 복호화 가능성을 제시한다.

이전 절에서 살펴본 바와 같이 AES 키와 HMAC 키는 Keystore에 저장된 암호키를 사용하여 암호화되어 저장된다. 하지만 암호화된 키들은 SAP Concur가 실행되는 중 복호화되어 메모리에 적재된다. 또한, AES 키와 HMAC 키에 대한 제로화 및 동적 할당 해제 동작을 수행하지 않아 메모리에서 평문 형태의 암호키를 추출할 수 있다.

실험은 다음과 같이 진행하였다. 실행 중인 SAP Concur 프로세스의 메모리를 추출한 후, 키워드 검

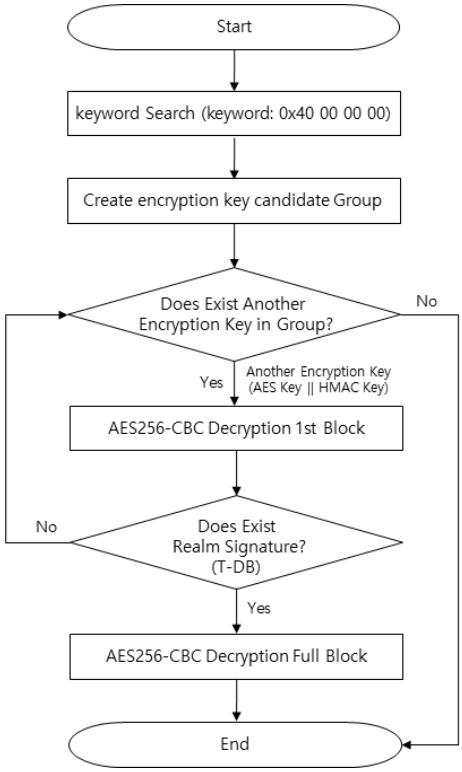

Fig. 6. Key recovery and database decryption process

색을 통해 암호키를 획득하였다. 그리고 메모리에서 획득한 암호키를 사용하여 데이터베이스 복호화를 진행하였다. SAP Concur의 Realm 데이터베이스 암호키 추출 및 복호화 동작 과정은 Fig. 6과 같다.

암호화에 사용된 AES 키와 HMAC 키는 Table 9와 같이 Blob 형태로 메모리에 적재된다. 상위 4바이트에는 암호키의 크기로 고정되며, 이후 AES 키와 HMAC 키

Table 9. Structure of Encryption Key in Memory

Offset	Size (Byte)	Content
0x00	4	Size of Encryption Key (0x40 00 00 00)
0x04	32	AES Key
0x24	32	HMAC Key

가 연접된다. 따라서 메모리 영역에서 고정된 16진수인 '40 00 00 00'을 키워드로 탐색을 진행하여 암호키 후보군을 생성할 수 있다.

암호키 후보군 생성 이후, 암호화된 데이터베이스의 첫 블록의 복호화를 진행하며, 복호화된 블록의 상위 0x10 위치에서 Realm 데이터베이스 시그니처인 'T-DB'의 존재 여부를 확인한다. 복호화된 첫 블록에 시그니처가 존재하지 않는다면 다음 암호키 후보군을 사용하여 복호화를 진행하며, Realm 시그니처 존재 여부를 확인한다. 만약 시그니처가 존재한다면 해당 암호키를 올바른 키로 추정하고 전체 데이터베이스 복호화를 진행한다. 암호키 재현을 통해 복호화된 데이터베이스는 Fig. 7과 같다.

V. 결 론

본 논문에서는 Android와 iOS에서 사용되는 Realm 데이터베이스의 구조 및 암·복호화 동작 과정에 대해서 분석하였다. Realm 데이터베이스는 AES256-CBC 암호화 알고리즘을 사용하여 데이터베이스를 암호화하며, HMAC-SHA224를 통해서 데이터의 무결성 검증 및 Fault Tolerance 기능을 수행함을 확인하였다. 또한, 실제 사용되는 SAP Concur 애플리케이션을 통해서 암호키 생성 과정을

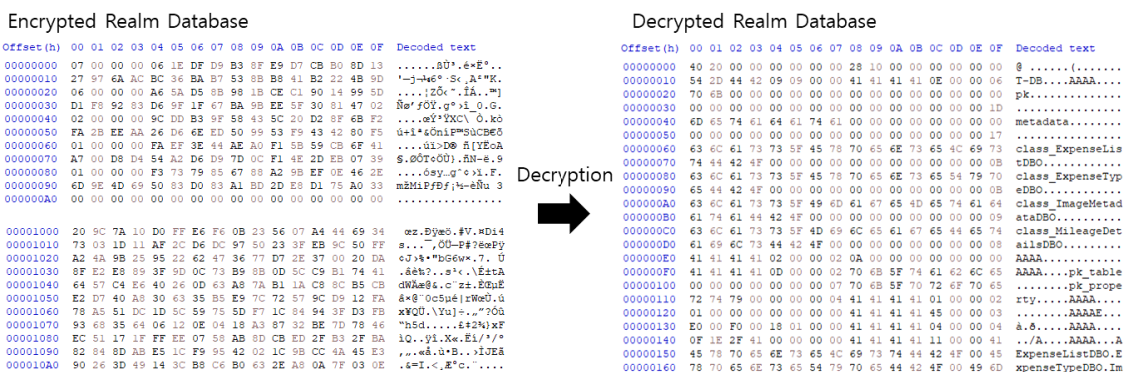

Fig. 7. Decryption of Encrypted Realm Database

분석하였으며, 이를 통해 암호화된 Realm 데이터베이스 복호화에 성공했다.

본 논문은 Realm 데이터베이스의 암·복호화 과정에 대한 첫 상세 분석 논문이다. 이를 활용한다면 암호화된 Realm 데이터베이스의 복호화를 통한 디지털 증거 확보가 가능하다. 현재 Realm 데이터베이스에 관한 연구는 활발히 이루어지고 있지 않지만, 사용률은 계속해서 증가하고 있으므로 본 연구가 향후 디지털포렌식 수사에 기여할 것으로 기대한다.

References

[1] "Realm blog", https://academy.realm.io/kr/posts/gdg-seoul-realm-introduce/

[2] "Realm blog", https://academy.realm.io/kr/posts/realm-object-centric-present-day-database-mobile-applications/

[3] "DB-Engine", https://db-engines.com/en/ranking

[4] Junki Kim, Jechyeok Han and Jong-Hyun Choi, "The Method of Recovery for Deleted Record of Realm Database," Jonornal of The Korea Institute of information Security & Cryptology, 28 (3), pp. 625-633, Jun. 2018

[5] J. Pu, Z. Song, and E. Tilevich, "Understanding the energy, performance, and programming effort trade-offs of Android persistence frameworks," 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 433-438, Sep. 2016

[6] Zhang, Lijun, and F. Yu. "The forensic analysis of WeChat message," 2016 IEE 6th International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC), pp. 500-503, Jul. 2016

[7] "Github", https://github.com/realm/realm-core

〈저자소개〉

윤 병 철 (Byungchul Youn) 학생회원
2019년 2월: 국민대학교 수학과 졸업
2019년 9월~현재: 국민대학교 금융정보보안학과 석사과정
〈관심분야〉 디지털 포렌식, 정보보호

박 명 서 (Myungseo Park) 학생회원
2013년 2월: 국민대학교 수학과 졸업
2015년 2월: 국민대학교 금융정보보안학과 석사
2014년 12월~2017년 2월: 국가보안연구소 연구원
2017년 3월~현재: 국민대학교 금융정보보안학과 박사과정
〈관심분야〉 디지털 포렌식, 암호 알고리즘, 정보보호

김 종 성 (Jongsung Kim) 종신회원
2000년 8월/2002년 8월: 고려대학교 수학 전공 학사/이학석사
2006년 11월: K.U.Leuven, ESAT/SCD-COSIC 정보보호 전공 공학박사
2007년 2월: 고려대학교 정보보호대학원 공학박사
2007년 3월~2009년 8월: 고려대학교 정보보호기술연구센터 연구교수
2009년 9월~2013년 2월: 경남대학교 e-비즈니스학과 조교수
2013년 3월~2017년 2월: 국민대학교 수학과 부교수
2014년 3월~현재: 국민대학교 일반대학원 금융정보보안학과 부교수
2017년 3월~현재: 국민대학교 정보보안암호수학과 부교수
〈관심분야〉 정보보호, 암호 알고리즘, 디지털 포렌식