닉스테크
서울시 성동구 왕십리로 58 포휴빌딩 10 층
전화 02-3497-8900 **팩스** 02-578-6621

NICS TECH
Network Integrated Computing Service Provider

CLOUD SECURITY BROKER
암호화키 관리 및 공유 시나리오

VERSION	**0.0.2**
DATE	**2016-08-01**
WRITER	**장욱**

1. 목차

2. 문서정보 및 수정내역

수정날짜	수정자	버전	추가/수정항목	내용
2016-08-01	장욱	0.0.1	초안 작성	초안 작성
2016-08-12	송천종	0.0.2	추가/수정	키 생성/관리, 데이터 업로드

3. 배경

목표

Cloud Security Broker (이하 CSB 라 칭함) 서비스에서 개인 문서를 클라우드 스토리지 저장 시 보안 강화를 위한 방법이 필요하다. 일반적인 경우 Saas 모델 서비스의 경우 (협업툴, 메신저등) 에서 개인 데이터는 마스터 키를 통한 암호화를 통해 서버에 저장된다. 하지만 유사시 마스터키가 유출 되면 서버내 모든 데이터가 유실 되는 휘험이 존재 한다. 따라서 CSB 은 서버에 데이터 저장 시 각 개인의 Private 키를 통한 암호화를 하는 것을 목표로 한다.

4. 키 관리 방식에 대한 문제

마스터키 방식

일반적인 Saas 모델 서비스 (협업 툴, 메신저) 등은 서버에 대화 원문, 혹은 개인 데이터 저장 이슈가 발생한다. 따라서 서버에 데이터 저장 시 (대화내용, 개인 데이터) 최소한의 보안을 위해 마스터키 방식을 통해 대화 저장하게 된다. 하지만 마스터 키 방식을 통할 경우 다음과 같은 문제점이 발생한다.

- ✓ 클라우드 서버 자체가 보안상이 이슈로 마스터키가 노출될 경우 -
  클라우드에 저장된 모든 사용자의 데이터가 유출 됨
- ✓ 개인 사용자의 데이터가 유출될 경우 -
  개인 사용자 데이터의 암호화 키를 유추/획득 시, 이를 이용한 서버내 모든 데이터가 유출됨

개인키 방식 (종단간 암호화 방식)

각 단말 (Terminal) 각 개인키를 가지고 데이터를 암호화 하는 방식은 보안상 유리한 점이 있으나 구현의 어려움이 따른다. 또한 CSB 서비스의 핵심 기능인 개인데이터를 외부에 공유할 경우 1:1 공유시는 문제 되지 않으나, 1:N 공유방식에는 구현이 복잡하다.

5. 변형된 개인 암호화 키 관리 방식

키 정의

- ✓ 데이터 키: 데이터 암/복호화 키
- ✓ 개인 키: 최초 사용자 인증 시 생성 (분실 시 데이터 암/복호화 불능)
- ✓ 고객사 마스터 키: 고객사 별 존재하는 암호화 키
- ✓ 암호화된 데이터 키: (데이터 키 XOR 개인키) 를 고객사 마스터키로 암호화한 키

전제

- ✓ HTTPS 통신을 전제로 한다.
- ✓ 개인 키는 최초 사용자 등록 시 클라이언트에서 생성되어 클라이언트 측에 보관 한다
- ✓ 고객사 마스터키는 서버 측에 관리되며 보안 사고에도 유출되지 않아야 한다. (HSM 등으로 관리)
- ✓ 데이터 암호화는 AES-256-CBC 를 사용 한다. (데이터 키 , 개인키는 32 byte)
- ✓ 데이터 키의 생성은 최초 사용자 인증시 서버 측에서 생성하며 이후 개인 키와 조합 처리되어 최종적으로 고객사 마스터키로 암호화 하여 클라이언트에 저장 하여 관리 한다.
- ✓ 데이터 키는 암 / 복호화 순간에 메모리에 잠시 존재하며 파일 또는 DB 에 저장 할 수 없다.
- ✓ 개인키 와 암호화된 데이터 키는 사용자 로그인시 마다 로그인 정보와 함께 암호화 키를 전달하는 방식으로 이루어 진다. 따라서 접속한 서버를 검증할 방법(Man in the middle Attack 에 대한 보완) 이 필요하다.
- ✓ HTTPS Man-in-the-middle Attack 은 클라이언트에서 서버의 인증서 검증 방식을 통해 보완한다.

장점

- ✓ 데이터 키는 메모리상에만 암/복호화시 잠시 존재하여 유출 위험이 적다.
- ✓ 클라우드 서버측 보안 사고 발생시에도 각 클라이언트의 키는 클라이언트가 관리 함으로서 데이터 복호화 불가
- ✓ 클라이언트 가 해킹을 당하더라도 서버의 마스터 키를 모르기 때문에 데이터 복호화가 불가능 하다.
- ✓ 서버의 마스터키를 획득한다고 해도 최악의 경우 해킹 당한 클라이언트 1 명의 데이터 외 나머지 고객의 데이터는 안전하다.

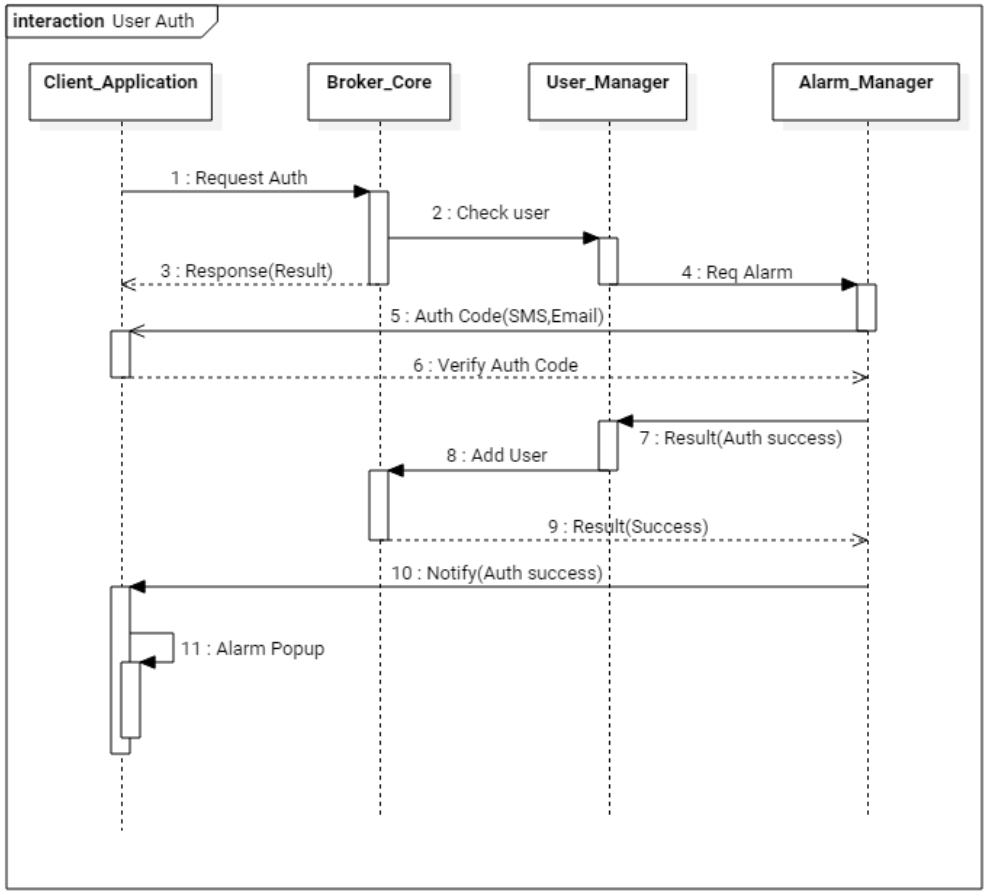
단점

- ✓ 중간에 사용자가 암호를 임의로 바꾸기 어렵다. (종단간 암호화도 동일함)
- ✓ 사용자가 개인키를 분실할 경우 개인키를 복구하기 힘듬

6. 시나리오 및 시퀀스

사용자 인증

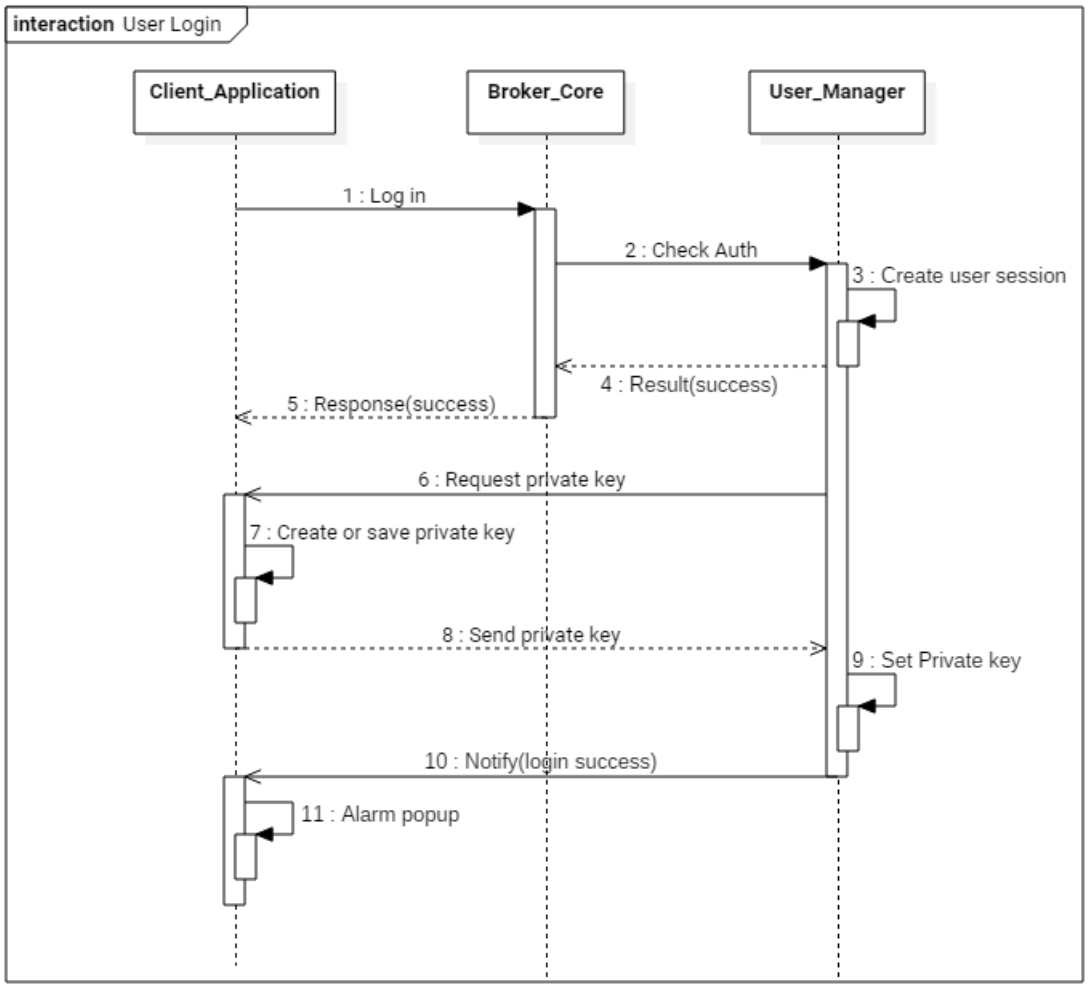
1) Client-Application 인증프로세스 진입후 Boker-Core 에게 사용자 인증을 요청한다.
2) Broker-Core 는 User-Manager 를 통해 사용자 유효성을 검사를 한다.
3) 사용자의 유효성에 문제가 없으면 Broker-System 은 User-Manager 에게 Alarm 요청을 한다 (인증코드).
4) Broker-Core 는 Bloking 되지 않고 Client-Applcation 에게 Response 를 보낸다.
5) Alarm-Manager 는 Client-Application 에게 인증코드 (Auth Code) 를 SMSS 혹은 Email 로 발송한다.
6) Client-Application 은 SMSS 나 Email 로 전달 받은 인증코드를 전송한다.
7) Alarm-Manager 는 인증 코드 유효성을 검사 후 문제가 없으면 User-Manage 에게 알린다.
8) User-Manager 는 사용자를 DB 에 등록하고 Broker-Core 에게 이를 통보한다.
9) Broker-Core 는 Alarm-Manager 에게 사용자 Notify (사용자 등록 완료) 를 요청한다.
10) Alarm-Manager 는 Client-Applcation 에 사용자 등록이 완료되었음을 알린다.
11) Client-Application 은 사용자에게 popup 등을 통해 사용자 인증이 완료 되었음을 통보한다



[그림 6-1 사용자 인증]

사용자 로그인

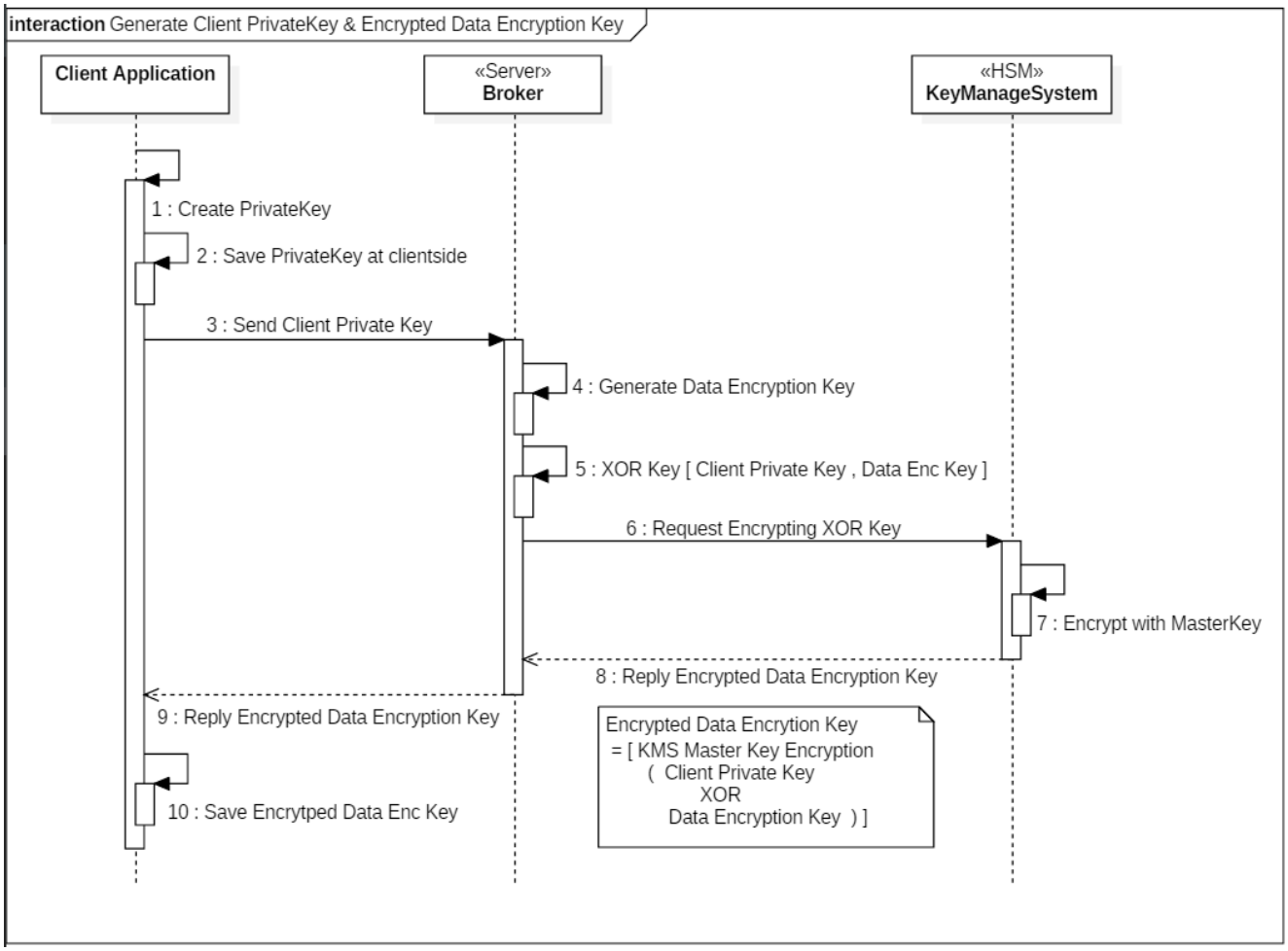
1) 사용자는 로그인을 시도 한다.
2) Broker-Core 는 로그인 User-Manger 를 통해 요청에 대해 유효성 검사를 요청한다.
3) 사용자 유효성에 문제가 없으면 User-Manger 는 User-Session 을 생성한다
4) User-Manager 결과를 Broker-Core 에게 통보한다.
5) Broker-Core 는 사용자에게 로그인 단계중 사용자 확인이 되었음을 통보한다.
6) User-Manager 는 Client-Application 에게 Private-Key 를 요청한다.
7) Client-Application 은 사용자 UI 등을 통해 Private-Key 를 생성 혹은 저장한다.
8) Client-Application 은 생성 혹은 저장한 Private-Key 를 User-Manager 에게 전달 한다.
9) User-Manage 는 생성된 User-Session 에 Private-Key 를 binding 한다.
10) User-Manager 는 결과를 Push 등을 통해 Client-Application 에 알린다.
11) Client-Application 은 최종 로그인 결과를 사용자에게 Popup 을 통해 알린다



[그림 6-2 사용자 로그인]

키 생성 (개인 키, 데이터(암호화 키), 암호화된 데이터 키)

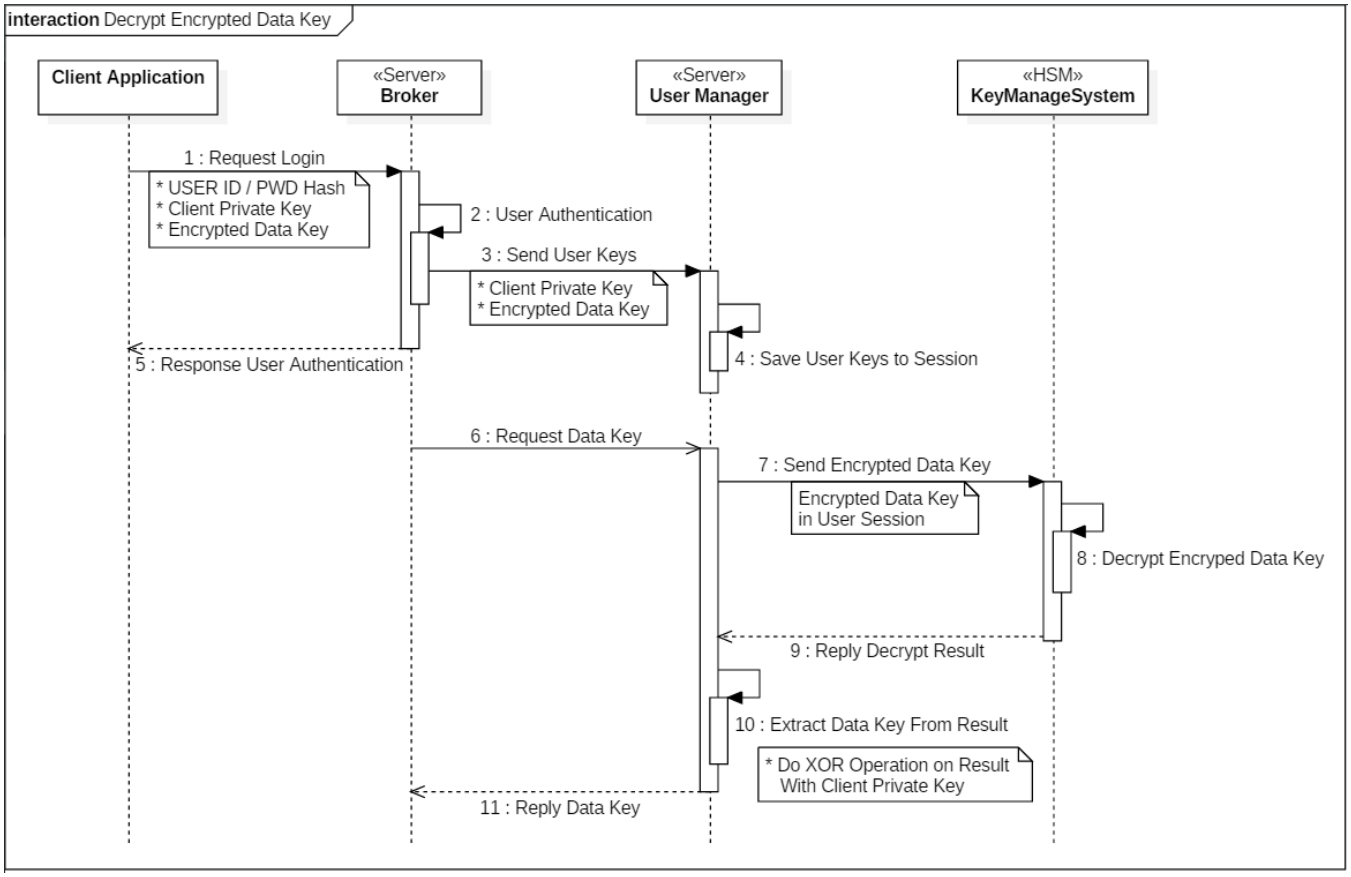
1) 사용자 최초 인증 시에 개인 키, 데이터 키, 암호화된 데이터 키를 생성 한다.
2) 개인키는 클라이언트 측에서 생성 및 관리한다.
3) 개인키, 데이터 키는 32 byte 랜덤키 형태로 구성(AES-256)
4) 데이터 키(암호화 키)는 서버에서 생성 하며, 별도 저장 및 관리 하지 않는다.
5) 데이터 키는 암/복호화 시에만 메모리에 잠시 존재 한 후 사라진다.
6) 암호화된 데이터 키는 데이터 키와 클라이언트의 개인 키를 XOR 연산한 후 서버 에서 마스터키로 암호화 하여 생성
7) 암호화된 데이터 키는 클라이언트로 전달되어 개인 키와 함께 클라이언트에서 저장 및 관리 한다.



[그림 6-3 개인키, 암호화된 데이터 암호화 키 생성]

암호화된 데이터 키 복호화

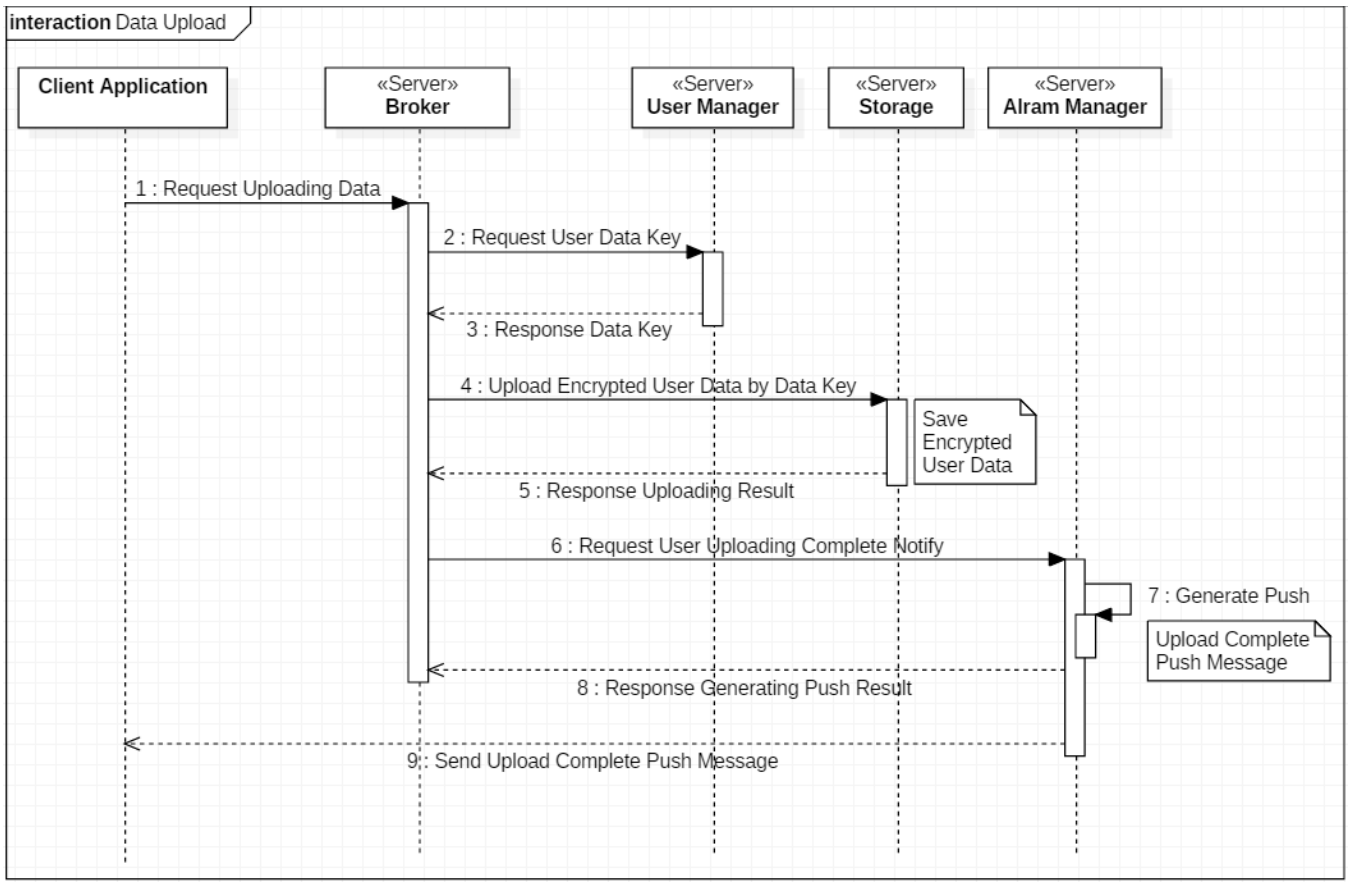
1) Client (Client-Application) 는 Broker-Core 에 로그인 한다
2) Broker-Core 는 사용자 인증 절차를 수행 하면서 사용자의 개인 키와 암호화된 데이터 키를 전송 받는다.
3) 인증이 성공 시 User-Manager 는 사용자의 개인 키와 암호화된 데이터 키를 넘겨받아 사용자 세션에 보존 한다.
4) User-Manager 는 사용자 데이터의 암/복호화 요청시 아래와 같은 과정을 수행하여 데이터 키를 넘겨준다
5) 암호화된 데이터 키를 사용자 고객사의 정보를 매개로 Key Management System 의 마스터 키를 통해 복호화 한다.
6) 복호화된 키에 사용자의 개인 키를 XOR 하여 데이터 키를 얻는다



[그림 6-4 암호화된 데이터 키 복호화]

데이터 업로드

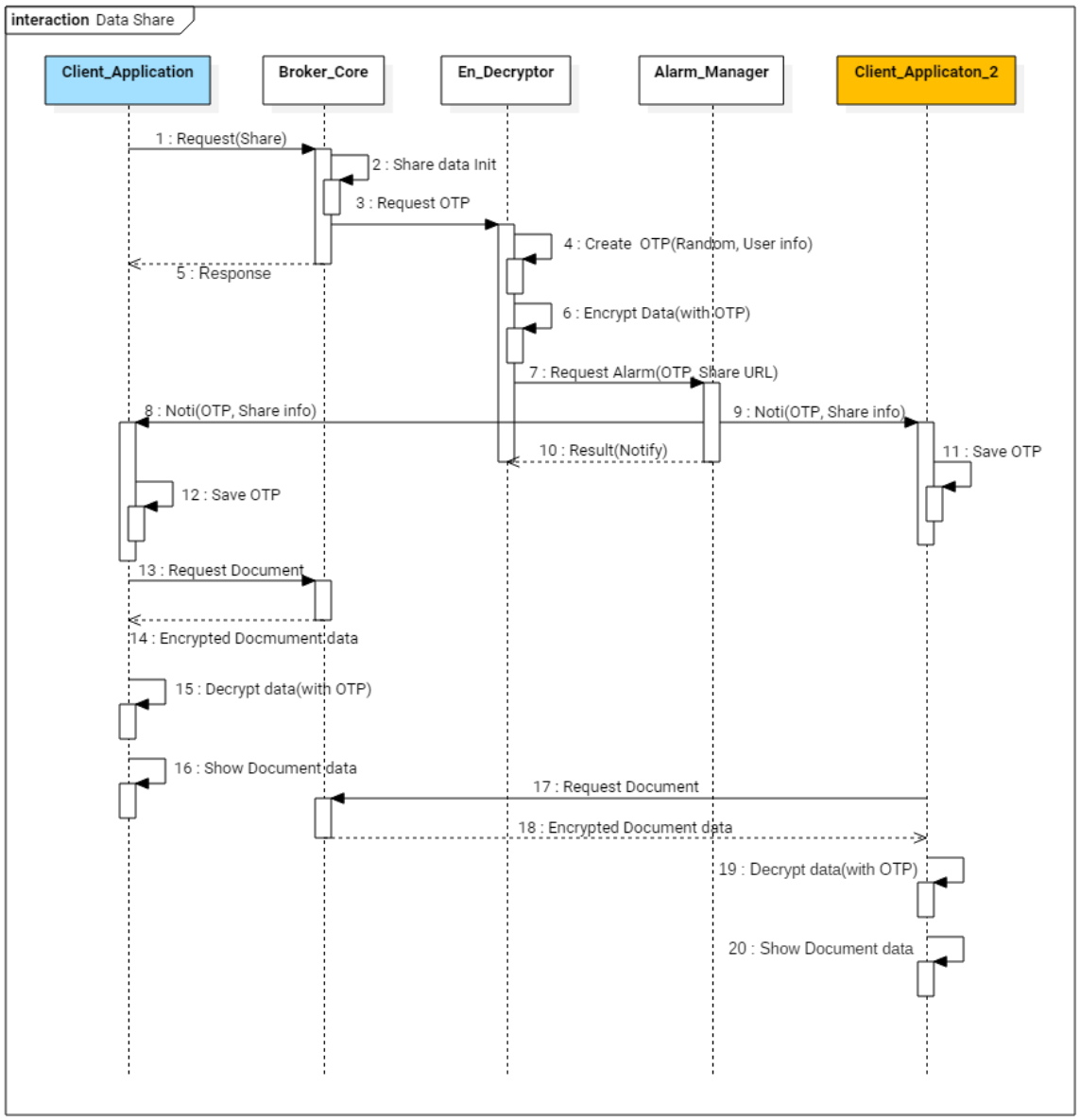
1) Client-Application 은 Broker-Core 에게 사용자 데이터 업로드를 요청 한다.
2) Broker-Core 는 User-Manager 를 통해 해당 사용자의 데이터 키를 요청한다.
3) User-Manager 는 데이터 키 복호화 과정을 통해 데이터 키를 Broker-Core 에 전달 한다.
4) Broker-Core 는 데이터 키로 사용자 데이터를 암호화하여 사용자가 지정한 저장소로 저장 한다.
5) Broker-Core 는 Alarm-Manager 에게 사용자 업로드 Notify 를 요청한다.
6) Alarm-Manager 는 데이터 저장 완료를 사용자에게 Push 를 통해 통보한다.
7) Client-Application 은 사용자에게 데이터가 저장 완료 되었음을 팝업 등을 통해 알린다.



[그림 6-5 사용자 데이터 업로드]

데이터 공유

1) Client-Application 가 문서 공유 요청을 한다.
2) Broker-Core 는 문서공유에 대한 정보를 초기화 한다 (문서 복호화, DB 저장등).
3) Broker-Core 는 En/Decryptor 모듈에게 OTP 생성을 요청한다.
4) En/Decryptor 는 랜덤값과 사용자 정보를 조합해 OTP 를 생성한다.
5) Broker-Core 는 Bloking 되지 않고 Client-Application 에게 Response 를 보낸다.
6) 생성된 OTP 로 공유할 문서를 암호화 한다.
7) 문서 암호화가 완료 되면 Alaram-Manager 에게 OTP 발송을 요청한다.
8) Alarm-Manager 는 공유자(공유하는 사람/Client-Application)에게 OTP 를 SMS 혹은 Push 로 발송한다.
9) Alarm-Manager 는 피공유자(공유받는 사람/Client-Application_2) 에게 OTP 를 SMS 혹은 Push 로 발송한다.
10) Alarm-Manager 는 OTP 발송 결과를 En/Decryptor 에게 리턴한다.
11) OTP 를 받은 피공유자(공유받는 사람/Client-Application_2) 는 수신한 OTP 와 문서정보를 로컬에 저장한다.
12) OTP 를 받은 공유자(공유하는 사람/Client-Application)는 수신한 OTP 와 문서정보를 로컬에 저장한다.
13) 공유자가 공유된 문서에 접근시 Client-Application 은 Broker-Core 문서 데이터를 요청한다.
14) Broker-Core 는 OTP 로 암호화된 데이터를 제공한다.
15) Client-Application 은 로컬에 저장된 OTP 와 문서정보를 통해 수신된 데이터를 복호화 한다.
16) Client-Application 은 복호화된 데이터를 화면에 표시한다.
17) 피공유자가 공유된 문서에 접근시 Client-Application_2 은 Broker-Core 문서 데이터를 요청한다.
18) Broker-Core 는 OTP 로 암호화된 데이터를 제공한다.
19) Client-Application_2 은 로컬에 저장된 OTP 와 문서정보를 통해 수신된 데이터를 복호화 한다.
20) Client-Application_2 은 복호화된 데이터를 화면에 표시한다.



[그림 6-6 사용자 데이터 공유]