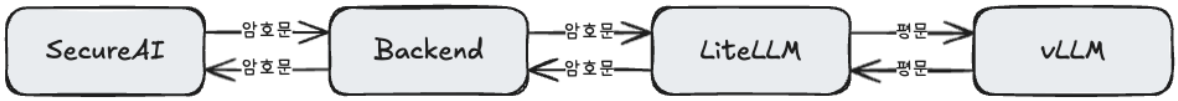
Server to Server 암호화 가이드

v1.0 | 2026-06-24 | 이니넥스트

SecureAI ↔ Backend ↔ LiteLLM 구간의 메시지를 INISAFE Net 으로 암호화하는 Server to Server 암호화 시스템의 연동 가이드이다.

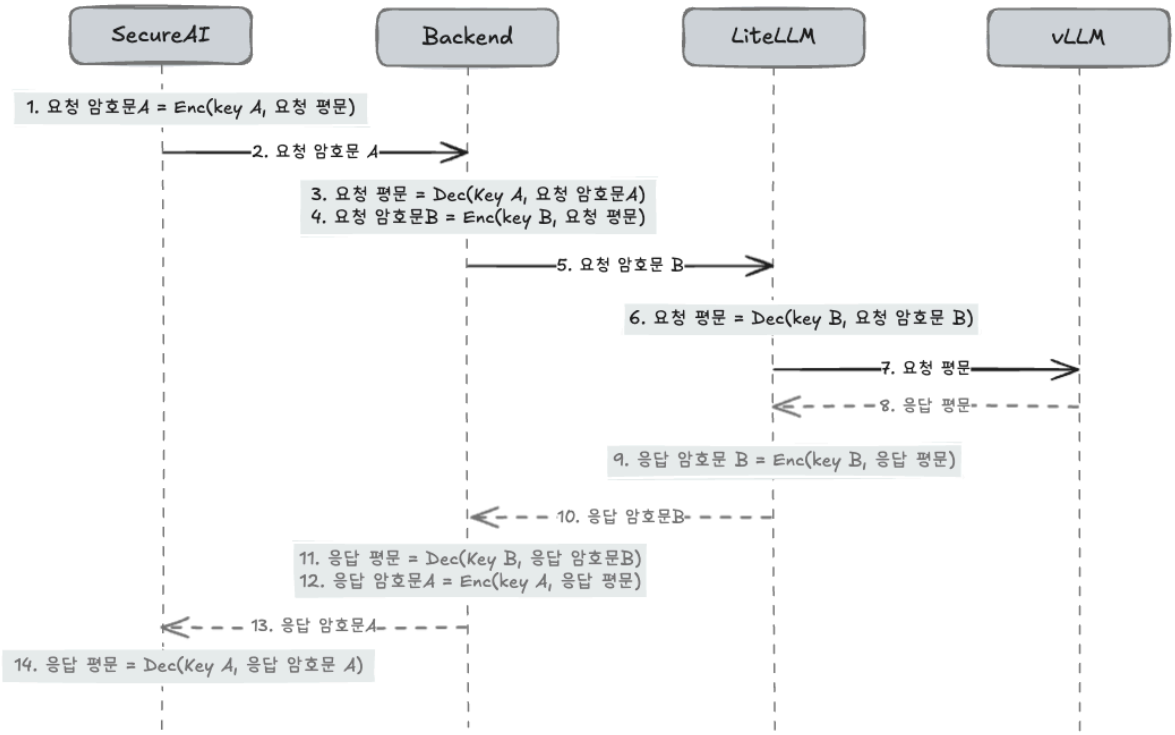
1. 개요

1.1 구성도



- SecureAI ↔ Backend 구간 : 요청과 응답을 INISAFE Net으로 암호화
- Backend ↔ LiteLLM 구간: 요청과 응답을 INISAFE Net으로 암호화
- LiteLLM ↔ vLLM 구간: 평문 (Plain Text)

1.2 암복호화 흐름



단계	위치	동작	설명
3	Backend	복호화	요청 암호문A를 Key A를 이용하여 복호화
4	Backend	암호화	요청 평문을 Key B를 이용하여 암호화
6	LiteLLM (pre_call_hook)	복호화	요청 암호문B를 Key B를 이용하여 복호화
9	LiteLLM (post_call_hook)	암호화	응답 평문을 Key B를 이용하여 암호화
11	Backend	복호화	응답 암호문B를 Key B를 이용하여 복호화
12	Backend	암호화	응답 평문을 Key A를 이용하여 암호화

1.3 배포 파일 구조

```
INISAFE_NET/
├── docs/
│   ├── Server To Server_암호화가이드.pdf
│   ├── INISAFE_NET_C_to_Python_v7.2_Developer_Guide_v1.0.pdf
│   └── INISAFE_NET_for_C_v7.2_Developer_Guide_v1.13.pdf
└── INISAFE_Net_for_C_v7.2.47_64_Linux_4.18.0.tar.gz # INISAFE Net SDK
```

- INISAFE Net SDK 상세 구조

```
INISAFE_Net_for_C_v7.2.47_64_Linux_4.18.0/
    ├── conf/
    │   ├── INISAFENet_A.cnf      # INISAFE 설정 파일 A
    │   └── INISAFENet_B.cnf      # INISAFE 설정 파일 B
    ├── keys/
    │   ├── EncryptKey_A.der      # 세션 키 파일 A
    │   └── EncryptKey_B.der      # 세션 키 파일 B
    ├── lib/
    │   ├── libINISAFE_Crypto_for_C_v5.2.0_Linux_4.18_64.so
    │   ├── libINISAFE_PKI_for_C_v5.2.6_Linux_4.18_64.so
    │   ├── libiniCore_v2.4.6_Linux_4.18.so
    │   └── libinisafeNet_7.2.47_Linux-4.18.0-193.28.1.el8_2.x86_64.so
    ├── sample/
    │   └── keyfix
    │       ├── inl_constants.py
    │       ├── inl_functions.py
    │       ├── keyfix.py         # 테스트 코드
    │       ├── keyfix_cnfkey.py  # 테스트 코드
    │       └── keyfix_dualkey.py # 다중키 테스트 코드
    ├── license/
    │   └── default.lic         # INISAFE 라이선스
    ├── log/
    └── tools/
        ├── gen_ivkey           # 세션 키 생성 도구
        ├── encrypt_pwd         # 비밀번호 암호화 도구
        └── decrypt_pwd         # 비밀번호 복호화 도구
```

2. 빠른시작

INISAFE Net SDK 동작을 빠르게 검증할 수 있는 최소 실행 절차이다.

사전 준비

- Docker 설치

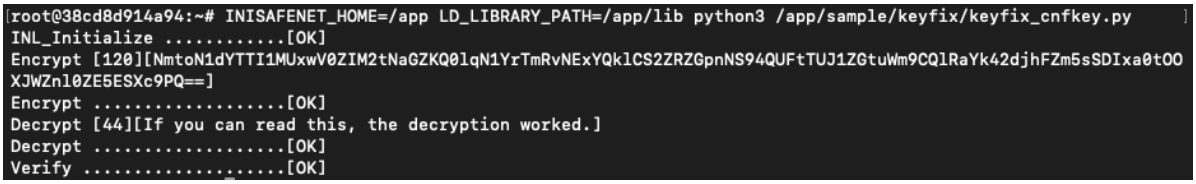
```
# 1. INISAFE Net 패키지 디렉토리로 이동
cd /path/to/INISAFE_Net_for_C_v7.2.47_64_Linux_4.18.0

# 2. Ubuntu 22.04 컨테이너 실행
#    현재 디렉토리를 컨테이너 내 /app으로 마운트
docker run -it -v "$(pwd):/app" --platform linux/amd64 ubuntu:22.04

# 3. Python 설치 (컨테이너 내부)
apt update && apt install -y python3

# 4. 스크립트 실행
INISAFENET_HOME=/app LD_LIBRARY_PATH=/app/lib python3
/app/sample/keyfix/keyfix_cnfkey.py
```

정상 동작 시 아래와 같이 출력된다.



3. SDK 설치

3.1 사전 요구사항

항목	요건
OS	Linux x86_64
INISAFE SDK	전달된 `INISAFE_Net_for_C_v7.2.47_64_Linux_4.18.0` 패키지

이하 `INISAFE_Net_for_C_v7.2.47_64_Linux_4.18.0` 패키지를 **inisafe 패키지**로 통칭한다.

3.2 SDK 배치 방법

> **Info**
>
> `inisafe/` 패키지 디렉토리를 Backend, LiteLLM 서버에서 접근할 수 있도록 배치한다. `INISAFENET_HOME` 과 `LD_LIBRARY_PATH` 만 올바르게 설정하면 된다.

방식 예시	설명	비고
서버별 복사	각 서버에 `inisafe/` 디렉토리를 복사	테스트 환경 권장
Docker 볼륨 마운트	호스트의 `inisafe/` 를 컨테이너에 마운트	

배치 후 아래 환경변수를 설정한다. (경로는 실제 배치 위치에 맞게 변경):

```
export INISAFENET_HOME=/path/to/inisafe
export LD_LIBRARY_PATH=$INISAFENET_HOME/lib:$LD_LIBRARY_PATH
```

Docker 볼륨 마운트 예시:

```
volumes:
  - /path/to/inisafe:/app/inisafe
environment:
  INISAFENET_HOME: /app/inisafe
  LD_LIBRARY_PATH: /app/inisafe/lib
```

3.3 심볼릭 링크 생성

> **Tip**
>
> `inisafe` 패키지 내에 이미 존재하는 경우 생략한다.

INISAFE 라이브러리는 **짧은 이름**으로 로딩이 필요하다. 실제 파일명이 버전을 포함하고 있으므로 심볼릭 링크를 생성한다.

```
cd inisafe/lib/

ln -sf libINISAFE_Crypto_for_C_v5.2.0_Linux_4.18_64.so    libiniCrypto.so
ln -sf libiniCore_v2.4.6_Linux_4.18.so                     libiniCore.so
ln -sf libINISAFE_PKI_for_C_v5.2.6_Linux_4.18_64.so       libiniPKI.so
ln -sf libinisafeNet_7.2.47_Linux-4.18.0-193.28.1.el8_2.x86_64.so
libinisafeNet.so
```

3.4 INISAFENet.cnf 설정

`inisafe/conf/INISAFENet.cnf` 파일에서 환경에 맞게 확인/수정해야 할 항목이다.

> **Note**
>
> 자세한 정보는 `INISAFE_NET_for_C_v7.2_Developer_Guide_v1.13.pdf` 파일의 `4. 환경설정 파일 설명`을 참고한다.

3.4.1 COMMON 섹션

항목	기본값	설명
`CM_LICENSE_PATH`	`$INISAFENET_HOME/license/license.lic`	라이선스 파일 경로
`CM_CRYPTO_ALG`	`SEED-CBC`	암호 알고리즘. 변경하지 않는 것을 권장
`CM_ENCODING_FLAG`	`0011`	BASE64 인코딩 활성화
`CM_LOG_LEVEL`	`ERROR`	디버깅 시 `DEBUG`로 변경. 운영 시 `ERROR` 권장
`CM_LOG_PATH`	`$INISAFENET_HOME/log`	로그 출력 경로

3.4.2 KEYFIX 섹션

> **Important**
>
> Backend와 LiteLLM 양쪽에서 **동일한 세션 키 파일과 비밀번호**를 사용해야 한다. 세션 키를 새로 발급해야 하는 경우 세션 키 생성 절을 참고한다.

항목	기본값	설명
`KF_SESSION_KEY_USE`	`Y`	세션 키 파일 자동 로드 활성화
`KF_SESSION_KEY_PATH`	`$INISAFENET_HOME/keys/EncryptKey.der`	세션 키 파일 경로
`KF_SESSION_KEY_PWD`	(암호화된 문자열)	세션 키 파일 복호화 비밀번호
`KF_PAD_LEN`	`16`	랜덤 패딩 크기 (v7.2.5 이상)

3.5 세션 키 생성

세션 키 재발급이 필요한 경우 아래 절차를 따른다.

전달된 `$INISAFENET_HOME/tools/gen_ivkey` 도구로 암호화된 세션 키 파일을 생성한다.

```
cd $INISAFENET_HOME/tools
./gen_ivkey
```

```
1: old format(use Hash)
2: new format(use PBKDF2)
3: new format(use PBKDF2,sha256)
Enter the create version: 1

< iv > Input the 16 chracter of the number,
Enter New iv : 1234567890123456      ← 16자리 IV 입력

< skey > Input the 16 chracter of the number,
Enter New skey : INISAFE NETWORK.    ← 16자리 세션 키 입력

Enter New Password : ********        ← 세션키 비밀번호 입력

Success to create session-key file: EncryptKey.der
```

3.5.1 encrypt_pwd로 비밀번호 암호화

`KF_SESSION_KEY_PWD`에 넣을 값은 `$INISAFENET_HOME/tools/encrypt_pwd` 도구로 생성한다.

```
cd $INISAFENET_HOME/tools
./encrypt_pwd
```

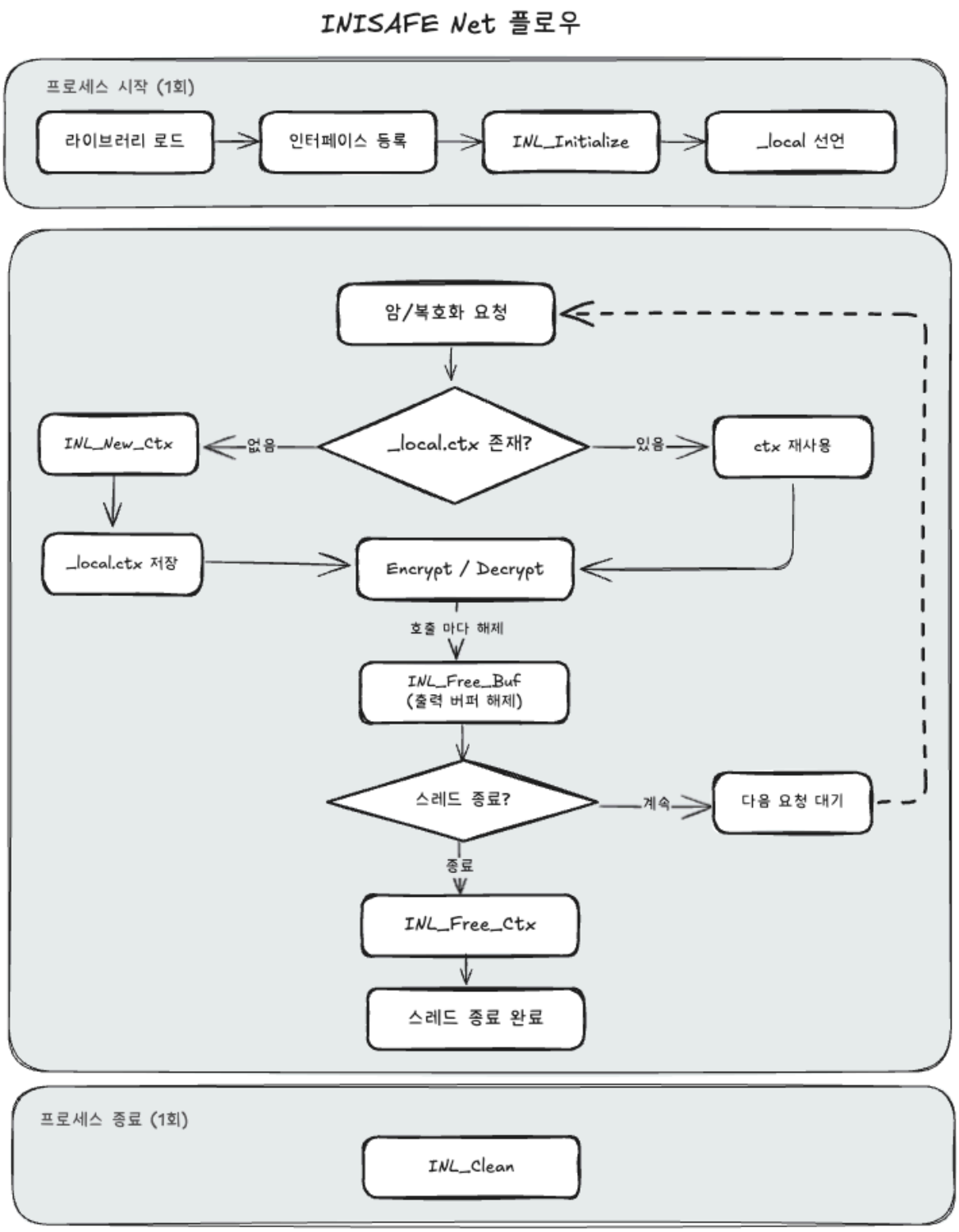
```
Enter Password : ********        ← 세션키 비밀번호 입력

Inputed Password : [********]
Encrypted Password : [QZcawOC1VDkgRljCFNYeU2EBysLOWtdWk52r+mHgRu8=]
```

생성된 `EncryptKey.der`를 `inisafe/keys/`에 배치하고, `INISAFENet.cnf`에서 `KF_SESSION_KEY_PATH`와 `KF_SESSION_KEY_PWD`를 수정한다.

4. 예제 및 사용법

> **Note**
>
> 자세한 연동 정보는 `INISAFE_NET_C_to_Python_v7.2_Developer_Guide_v1.0.pdf` 파일을 참고한다.


INISAFE Net 플로우
프로세스 시작 (1회)
라이브러리 로드
인터페이스 등록
INL_Initialize
_local 선언
암/복호화 요청
_local.ctx 존재?
없음
INL_New_Ctx
있음
ctx 재사용
_local.ctx 저장
Encrypt / Decrypt
호출 마다 해제
INL_Free_Buf
(출력 버퍼 해제)
스레드 종료?
계속
다음 요청 대기
종료
INL_Free_Ctx
스레드 종료 완료
프로세스 종료 (1회)
INL_Clean

4.1 라이브러리 로드

의존 라이브러리(Crypto, Core, PKI)를 메인 라이브러리(Net)보다 먼저 RTLD_GLOBAL 옵션으로 로드해야 한다.

```
import ctypes
import threading
import base64
import os
import atexit
import weakref

# 의존성 순서대로 로딩
ctypes.CDLL("libiniCrypto.so", mode=ctypes.RTLD_GLOBAL)
ctypes.CDLL("libiniCore.so", mode=ctypes.RTLD_GLOBAL)
ctypes.CDLL("libiniPKI.so", mode=ctypes.RTLD_GLOBAL)
lib = ctypes.CDLL("libinisafeNet.so")
```

4.2 인터페이스 등록

포인터를 사용하는 함수 호출 시 메모리 오류를 방지하기 위해 함수 시그니처를 등록한다.

```
lib.INL_Initialize.argtypes = [ctypes.c_int, ctypes.c_char_p, ctypes.c_char_p]
lib.INL_Initialize.restype = ctypes.c_int
lib.INL_New_Ctx.argtypes = [ctypes.c_int, ctypes.POINTER(ctypes.c_void_p)]
lib.INL_New_Ctx.restype = ctypes.c_int
lib.INL_Free_Ctx.argtypes = [ctypes.c_void_p]
lib.INL_Free_Ctx.restype = ctypes.c_int
lib.INL_Encrypt.argtypes = [
    ctypes.c_void_p, ctypes.c_char_p, ctypes.c_int,
    ctypes.POINTER(ctypes.c_char_p), ctypes.POINTER(ctypes.c_int),
]
lib.INL_Encrypt.restype = ctypes.c_int
lib.INL_Decrypt.argtypes = [
    ctypes.c_void_p, ctypes.c_char_p, ctypes.c_int,
    ctypes.POINTER(ctypes.c_char_p), ctypes.POINTER(ctypes.c_int),
]
lib.INL_Decrypt.restype = ctypes.c_int
lib.INL_Free_Buf.argtypes = [ctypes.c_char_p]
lib.INL_Free_Buf.restype = None
lib.INL_Clean.argtypes = []
lib.INL_Clean.restype = None
lib.INL_Error_String.argtypes = [ctypes.c_int]
lib.INL_Error_String.restype = ctypes.c_char_p
```

4.3 초기화

모듈 초기화, 프로세스 시작 시 1회만 호출한다.

```
FIXKEY_CTX = 0x08
conf_path = os.path.join(os.environ["INISAFENET_HOME"], "conf",
"INISAFENet.cnf")

ret = lib.INL_Initialize(FIXKEY_CTX, conf_path.encode("utf-8"), None)
if ret != 0:
    err = lib.INL_Error_String(ret)
    err_msg = err.decode("utf-8", errors="replace") if err else "unknown"
    raise RuntimeError(f"INL_Initialize failed: code={ret}, {err_msg}")

# 프로세스 종료 시 자동으로 리소스 정리
atexit.register(lib.INL_Clean)
```

4.4 ctx 초기화

스레드당 최초 1회만 컨텍스트를 생성하고 이후 재사용한다. 워커 스레드 종료 시 weakref를 통해 INL_Free_Ctx가 자동 호출되며, 메인 스레드는 atexit의 INL_Clean이 처리한다.

```
_local = threading.local()

def _free_ctx(ctx: ctypes.c_void_p) -> None:
    if ctx.value is not None:
        lib.INL_Free_Ctx(ctx)
        ctx.value = None

def _get_ctx() -> ctypes.c_void_p:
    if not hasattr(_local, "ctx"):
        ctx = ctypes.c_void_p(None)
        ret = lib.INL_New_Ctx(FIXKEY_CTX, ctypes.byref(ctx))
        if ret != 0:
            raise RuntimeError(f"INL_New_Ctx failed: {ret}")
        _local.ctx = ctx

        # 메인 스레드는 atexit(INL_Clean)이 처리하므로 제외
        if threading.current_thread() is not threading.main_thread():
            weakref.finalize(
                threading.current_thread(),
                _free_ctx,
                ctx,
            )
    return _local.ctx
```

4.5 암호화

평문 bytes를 INL_Encrypt로 암호화하여 암호문 bytes 반환한다.

```
def encrypt(plaintext: bytes) -> bytes:

    ctx  = _get_ctx()
    out  = ctypes.c_char_p(None)
    outl = ctypes.c_int(0)

    try:
        ret = lib.INL_Encrypt(ctx, plaintext, len(plaintext),
                              ctypes.byref(out), ctypes.byref(outl))
        if ret != 0:
            raise RuntimeError(f"INL_Encrypt failed: {ret}")

        return out.value[:outl.value]
    finally:
        if out:
            lib.INL_Free_Buf(out)
```

4.6 복호화

암호문 bytes를 INL_Decrypt로 복호화하여 평문 bytes 반환한다.

```
def decrypt(ciphertext: bytes) -> bytes:

    ctx  = _get_ctx()
    out  = ctypes.c_char_p(None)
    outl = ctypes.c_int(0)

    try:
        ret = lib.INL_Decrypt(ctx, ciphertext, len(ciphertext),
                              ctypes.byref(out), ctypes.byref(outl))
        if ret != 0:
            raise RuntimeError(f"INL_Decrypt failed: {ret}")

        return out.value[:outl.value]
    finally:
        if out:
            lib.INL_Free_Buf(out)
```

5. Backend 연동 방식 (다중 키 사용 암호화)

SecureAI ↔ Backend 구간과 Backend ↔ LiteLLM 구간에 서로 다른 세션 키를 사용하는 경우의 구현 방법을 설명한다.

5.1 구조

Backend(python 서버)가 Key A와 Key B를 동시에 보유하여 구간별로 다른 키로 암복호화를 처리한다.

단계	구간	동작
① 요청 수신	SecureAI → Backend	`X-Sgt-Upstream-Enc: true` 헤더 확인 후 Key A로 복호화
② 요청 송신	Backend → LiteLLM	Key B로 요청을 암호화 후 전송
③ 응답 수신	LiteLLM → Backend	응답을 Key B로 복호화
④ 응답 송신	Backend → SecureAI	응답을 Key A로 암호화 + `X-Sgt-Upstream-Enc: true`

> **Note**
>
> SDK는 이중 세션 기능을 별도로 제공하지 않는다. 업무단(애플리케이션)에서 키 변환을 직접 처리해야 한다.

5.2 SecureAI ↔ Backend 연동 시 준수사항

단계	구간	동작	구현 주체
①	SecureAI → Backend	`base64(encrypt(평문 JSON))` 전송, 헤더 `X-Sgt-Upstream-Enc: true`	이니넥스트 (기구현)
②	Backend (요청 복호화)	base64 decode → `decrypt()` → 평문 JSON 복원	**고객사**
③	Backend (응답 암호화)	`encrypt()` → `base64(암호 byte[])`	**고객사**
④	Backend → SecureAI	base64 암호 응답 전송 + 헤더 echo	**고객사**
⑤	SecureAI (응답 복호화)	base64 decode → `decrypt()` → 평문	이니넥스트 (기구현)

1. **요청 복호화** — `X-Sgt-Upstream-Enc: true`가 붙은 요청 본문을 base64 decode → `decrypt()`로 평문 JSON 복원
2. **응답 암호화 (비스트림)** — LLM 응답 본문 전체를 `encrypt()` 후 base64 인코딩 + 헤더 echo
3. **응답 암호화 (스트림)** — SSE/NDJSON 프로토콜 구조는 그대로 유지하고, payload만 `base64(cipher)`로 치환

#	항목	규칙	비고
C1	식별 헤더	`X-Sgt-Upstream-Enc: true`	요청에 있으면 암호 요청. 응답에도 동일 헤더를 echo 해야 게이트웨이가 복호화
C2	base64 전송 인코딩	`encrypt(byte[])` 출력을 개행 없는 base64로 인코딩해 바디로 전송	JSON 래핑 금지. 표준 알파벳(URL-safe 아님), 디코드도 표준 디코더 사용
C3	바이트로 다룰 것	base64 decode 후 `new String(...)` 변환 전에 `byte[]` 단계에서 복호화	암호 바이트는 UTF-8 왕복 시 손상됨
C4	동일 키	게이트웨이와 같은 `EncryptKey.der` (또는 같은 session-key)	`EncryptedSkeyPath` 또는 `SGT_UPSTREAM_SESSION_KEY` 로 지정
C5	동일 EncodingFlag	`CM_ENCODING_FLAG=0010` (HEX)	`0011` 사용 시 HTTP 전송 중 깨짐. 전송 base64(C2)와 별개 설정
C6	동일 알고리즘 설정	`CM_CRYPTO_ALG` 등 게이트웨이와 동일하게 맞출 것	게이트웨이의 `INISAFENet.properties` 를 그대로 복사하는 것이 가장 안전
C7	압축 금지	암호 바디에 gzip/deflate 적용 금지	게이트웨이는 요청에서 `Accept-Encoding` 을 제거해 보냄
C8	비스트림 Content-Length 필수	비스트림 암호 응답은 base64 바디 길이로 `Content-Length` 반드시 설정	게이트웨이는 암호문에 JSON 완결 검사를 못 해 CL로만 완결 판정
C9	스트림 프레이밍	SSE: `data: {base64(cipher)}\n\n` / NDJSON: `{base64(cipher)}\n`	`event:` / `id:` 등 다른 줄은 유지. payload(base64) ≤ 1 MiB
C10	Fail-closed	헤더=true인데 복호화 불가/실패면 400 반환, 평문 폴백 금지	에러 바디는 평문·비-2xx·헤더 미echo → 게이트웨이가 복호화 시도 안 함
C11	스트림 Content-Type 보존	스트림 응답은 `text/event-stream` / `application/x-ndjson` 유지	게이트웨이가 Content-Type으로 SSE/NDJSON 분기 및 경계 판정

5.2.1 SSE (`text/event-stream`) 연동 예시

프로토콜 래퍼(`data:` , 빈 줄, `\n\n` 종결)는 평문과 동일하다. **`data:` 뒤 payload만** `base64(cipher)` 로 치환한다.

```
# 평문
data: hi~

data: how

data: r u?
```

```
# 암호문 (실제 base64 값은 KeyFix 결과에 따라 달라진다)
data: AQpofiE=

data: AQhow==

data: AXIgdT8=
```

multi-line event의 경우 `event:` / `id:` 줄은 그대로 두고, `data:` 줄 payload만 치환한다.

```
event: message
data: {base64(cipher)}
```

5.2.2 SSE (`application/x-ndjson`) 연동 예시

한 줄의 **본문**(trailing `\n` 제외)만 `base64(cipher)` 로 치환한다.

```
# 평문
{"content":"hi"}\n
{"content":"there"}\n

# 암호문
{base64(cipher)}\n
{base64(cipher)}\n
```

5.3 초기화

라이브러리 로드와 인터페이스 등록은 라이브러리 로드, 인터페이스 등록 절을 참고한다.

초기화는 앞의 초기화 절과 동일하나 conf 경로를 Key A, Key B 각각 선언한다. 두 conf 파일은 동일한 `INISAFENET_HOME/conf/` 디렉토리에 위치한다.

```
INISAFENET_HOME = os.environ["INISAFENET_HOME"]
conf_a = os.path.join(INISAFENET_HOME, "conf", "INISAFENet_A.cnf")  # Key A
conf_b = os.path.join(INISAFENET_HOME, "conf", "INISAFENet_B.cnf")  # Key B

# INL_Initialize 호출은 ctx 초기화에서 수행한다.
```

5.4 ctx 초기화

앞의 ctx 초기화 절의 `_get_ctx()`를 Key A, Key B로 분리한다. `_new_ctx`는 `INL_Initialize` → `INL_New_Ctx`를 lock 안에서 원자적으로 실행하여 스레드 간 키 혼용을 방지한다.

```
_init_lock         = threading.Lock()
_local             = threading.local()
_atexit_registered = False

def _free_ctx(ctx: ctypes.c_void_p) -> None:
    if ctx.value is not None:
        lib.INL_Free_Ctx(ctx)
        ctx.value = None

def _new_ctx(conf_path: str) -> ctypes.c_void_p:
    global _atexit_registered
    with _init_lock:
        ret = lib.INL_Initialize(FIXKEY_CTX, conf_path.encode("utf-8"), None)
        if ret != 0:
            err = lib.INL_Error_String(ret)
            err_msg = err.decode("utf-8", errors="replace") if err else
"unknown"
            raise RuntimeError(f"INL_Initialize failed: code={ret},
{err_msg}")

        ctx = ctypes.c_void_p(None)
        ret = lib.INL_New_Ctx(FIXKEY_CTX, ctypes.byref(ctx))
        if ret != 0:
            err = lib.INL_Error_String(ret)
            err_msg = err.decode("utf-8", errors="replace") if err else
"unknown"
            raise RuntimeError(f"INL_New_Ctx failed: code={ret}, {err_msg}")

        if not _atexit_registered:
            atexit.register(lib.INL_Clean)
            _atexit_registered = True
    return ctx

def _get_ctx_a() -> ctypes.c_void_p:
    if not hasattr(_local, "ctx_a"):
        _local.ctx_a = _new_ctx(conf_a)
        if threading.current_thread() is not threading.main_thread():
            weakref.finalize(threading.current_thread(), _free_ctx,
_local.ctx_a)
    return _local.ctx_a

def _get_ctx_b() -> ctypes.c_void_p:
    if not hasattr(_local, "ctx_b"):
        _local.ctx_b = _new_ctx(conf_b)
        if threading.current_thread() is not threading.main_thread():
            weakref.finalize(threading.current_thread(), _free_ctx,
_local.ctx_b)
    return _local.ctx_b
```

5.5 암호화 / 복호화

앞의 암호화, 복호화 절과 동일하나 ctx를 인자로 받아 Key A / Key B 모두에서 재사용할 수 있도록 변경한다.

```
def encrypt(ctx: ctypes.c_void_p, plaintext: bytes) -> bytes:

    out  = ctypes.c_char_p(None)
    outl = ctypes.c_int(0)
    try:
        ret = lib.INL_Encrypt(ctx, plaintext, len(plaintext),
                              ctypes.byref(out), ctypes.byref(outl))
        if ret != 0:
            raise RuntimeError(f"INL_Encrypt failed: {ret}")

        return out.value[:outl.value]
    finally:
        if out:
            lib.INL_Free_Buf(out)

def decrypt(ctx: ctypes.c_void_p, ciphertext: bytes) -> bytes:

    out  = ctypes.c_char_p(None)
    outl = ctypes.c_int(0)
    try:
        ret = lib.INL_Decrypt(ctx, ciphertext, len(ciphertext),
                              ctypes.byref(out), ctypes.byref(outl))
        if ret != 0:
            raise RuntimeError(f"INL_Decrypt failed: {ret}")

        return out.value[:outl.value]
    finally:
        if out:
            lib.INL_Free_Buf(out)
```

6. LiteLLM Hook 연동 방식

LiteLLM Proxy는 LLM API 호출 전후에 데이터를 수정할 수 있는 Call Hook을 제공한다. INISAFE Net 암·복호화는 `CustomLogger`를 상속한 Hook 클래스로 구현하는 방식을 권장하나, Hook 외 다른 방식으로 구현해도 무방하다.

참고: https://docs.litellm.ai/docs/proxy/call_hooks

6.1 파일 구성

LiteLLM Hook 로직 파일과 INISAFE Net 모듈을 분리하여 작성한다.

```
├── litellm_config.yaml # LiteLLM 프록시 설정
├── my_hook.py          # LiteLLM Hook 로직 (CustomLogger 구현)
└── inisafe.py          # INISAFE Net 초기화 + 암복호화 (예제 및 사용법 참고)
```

`inisafe.py`는 예제 및 사용법 장의 코드를 그대로 사용한다. `my_hook.py`에서 `encrypt`, `decrypt`만 import하여 사용한다.

6.2 Hook 구현 (my_hook.py)

`CustomLogger`를 상속하여 아래 3개 메서드를 구현한다.

hook	시점	동작
`async_pre_call_hook`	LLM 호출 직전	요청 메시지 content 복호화
`async_post_call_success_hook`	LLM 응답 수신 후	응답 메시지 content 암호화
`async_post_call_streaming_iterator_hook`	SSE 스트리밍 중	각 chunk delta content 암호화

```python
# my_hook.py
from typing import Any, AsyncGenerator, Literal
from litellm.integrations.custom_logger import CustomLogger
from litellm.proxy.proxy_server import UserAPIKeyAuth, DualCache
from litellm.types.utils import ModelResponseStream

from inisafe import encrypt, decrypt  # 예제 및 사용법에서 작성한 inisafe.py


class INISAFENetHook(CustomLogger):

    # LLM 호출 직전
    async def async_pre_call_hook(
        self,
        user_api_key_dict: UserAPIKeyAuth,
        cache: DualCache,
        data: dict,
        call_type: Literal[
            "completion", "text_completion", "embeddings",
            "image_generation", "moderation", "audio_transcription",
        ],
    ):
        # 요청 메시지의 content를 복호화하여 LLM에 평문으로 전달
        for msg in data.get("messages", []):
            if isinstance(msg.get("content"), str):
                msg["content"] = decrypt(msg["content"])
        return data


    # LLM 응답 수신 후
    async def async_post_call_success_hook(
        self,
        data: dict,
        user_api_key_dict: UserAPIKeyAuth,
        response,
    ):
        # LLM 응답 content를 암호화하여 클라이언트에 반환 (non-streaming)
        for choice in getattr(response, "choices", []):
            msg = getattr(choice, "message", None)
            if msg and isinstance(getattr(msg, "content", None), str):
                msg.content = encrypt(msg.content)
        return response


    # SSE 스트리밍 중
    async def async_post_call_streaming_iterator_hook(
        self,
        user_api_key_dict: UserAPIKeyAuth,
        response: Any,
        request_data: dict,
    ) -> AsyncGenerator[ModelResponseStream, None]:
        # 스트리밍 응답의 각 chunk를 암호화하여 클라이언트에 반환
```

```
asyncforchunkinresponse:forchoiceingetattr(chunk,"choices",
[]):delta=getattr(choice,"delta",None)ifdeltaandisinstance(getattr(delta,"cont
ent",None),str):delta.content=encrypt(delta.content)yieldchunkproxy_handler_in
stance=INISAFENetHook()
```

6.3 litellm_config.yaml 등록

`my_hook.py` 를 `callbacks` 에 `파일명.인스턴스변수명` 형식으로 등록한다.

```
litellm_settings:
  callbacks: ["my_hook.proxy_handler_instance"]
```

7. 연동 시 유의사항

7.1 준수 사항

- SecureAI ↔ Backend 구간 : EncryptKey_A.der 키로 암/복호화 해야 한다.
- Backend ↔ LiteLLM 구간: EncryptKey_B.der 키로 암/복호화 해야 한다.
- `INL_Free_Ctx` 와 `INL_Free_Buf` 는 매 호출 시 **반드시** 수행해야 한다. (메모리 누수 방지)
- `INL_Clean` 은 프로세스 종료 시 1회 호출한다. (`atexit` 등록 권장)
- `$INISAFENET_HOME/log/` 디렉토리가 반드시 존재해야 한다.

7.2 문제 해결

증상	원인	해결
`Fail to Init log` (code 9091)	`$INISAFENET_HOME/log/` 디렉토리 없음	`inisafe/log/` 디렉토리 생성
`Invalid License` (code 8016)	라이선스 파일 없거나 잘못된 파일	`INISAFENet.cnf` 의 `CM_LICENSE_PATH` 확인
`Fail to decrypt data` (code 3007)	세션 키 파일 누락 또는 불일치	`KF_SESSION_KEY_USE=Y` 확인, 키 파일 경로 및 비밀번호 확인
`libiniCrypto.so: cannot open`	`LD_LIBRARY_PATH` 에 lib 경로 미포함	`export LD_LIBRARY_PATH=..`
`INL_Encrypt/Decrypt failed`	세션 키 불일치 또는 데이터 손상	구간별 세션키 확인