



xFile RAG-Guard

정부, 국방, 금융 등 중요정보 환경을 위한 RAG·LLM 보안 암호화 솔루션

기밀성 · 다중 테넌트 키 격리 · 검색 성능 95% 이상 유지

AES-256 호환 강도 · 동형암호 대비 자원 효율 · 국산 Gateway 연동

목 차

본 제안서의 구성

01	제안 배경	LLM·RAG 도입 시 보안 과제
02	RAG-Guard 솔루션 개요	4 계층 암호화 보안 레이어
03	핵심 기능 ① 프롬프트·벡터 암호화	SRXT 기반 유사도 보존 변환
04	핵심 기능 ② 문서 원장 보관 암호화	AES-256, RFC 3394 키 관리
05	핵심 기능 ③ BM25 전문검색 암호화	HMAC 결정적 토큰 방식
06	핵심 기능 ④ 삼중 방어 구조	BM25·KNN·평문 보호
07	배포 구조 및 국산 제품 호환	API Gateway / 벡터·그래프 DB
08	계층별 암호화 지원 범위	종합 적용 매트릭스
09	차별점 및 도입 효과	동형암호 대비 강점
10	연락처 및 맺음말	문의 및 후속 진행

정부, 국방, 금융 등 중요정보 (CSO 분류) 환경의 LLM·RAG 도입 과제

민감정보 유출 위험 없이 생성형 AI 의 생산성을 활용하기 위한 보안 요구

ISSUE 01

원문 노출 위험

RAG 구조상 청크 원문이 벡터 DB / 검색 인덱스에 평문 저장되어, 관리자·운영자 단위에서 열람 가능

ISSUE 02

벡터 역추론

임베딩 벡터로부터 원문 재구성 공격이 보고되고 있어, 벡터 저장 자체가 기밀 유출 경로가 될 수 있음

ISSUE 03

전문검색 양립 불가

일반 블록 암호화 적용 시 BM25·형태소 분석기와 병행이 불가, 기능과 보안의 동시 확보가 어려움

본 제안의 목적

RAG-Guard 는 위 세 가지 과제를 동시에 해결하기 위해 설계된 전용 보안 패키지입니다 .

① 벡터·원장 텍스트 전 구간 암호화 ② 검색 성능 95% 이상 유지 ③ 조직·테넌트별 완전 격리 ④ 국산 인프라와 호환

RAG-Guard 보안 레이어 개요

LLM 모델에 독립적인 플러그인형 보안 미들웨어

독립적 보안 미들웨어 구조

RAG-Guard 는 특정 LLM·벡터 DB 제품에 종속되지 않는 암호화 레이어입니다 . 국산 LLM 과 국산 검색 엔진에도 동일한 강도로 적용할 수 있도록 설계되었습니다 .

보안 강도	AES-256 비트 호환
키 관리 표준	RFC 3394 기반 계층적 키 생성
검색 성능	원본 대비 95% 이상 유지
다중 테넌트	조직단위 분리 키 구조
자원 요구	GPU 불필요 , CPU 연산만으로 처리
배포 형태	Python 패키지 / Docker / TEE 서비스

4 계층 보안 레이어

L1	프롬프트 / 쿼리 벡터 암호화 SRXT 유사도 보존 차원 최적화 변환 기술
L2	학습 시 문서 임베딩 벡터 암호화 SRXT + 분산 유도 키 관리
L3	학습 원장 · 청크 텍스트 암호화 AES-256, RFC 3394 키 래핑
L4	전문검색 토큰 암호화 HMAC-SHA256 결정적 토큰 (Blind Index)

프롬프트 · 문서 벡터 암호화 (SRXT)

유사도 보존 차원최적화 독자 알고리즘 기반 벡터 보호 (차원보존 및 분산환경에 적합하도록 벡터차원 최적화 기술)

적용 범위

- RAG 사용자 프롬프트 벡터화 시점 암호화
- 학습 시점 문서 (인덱스) 임베딩 벡터 암호화
- 조직별 분리 키 구조 — 테넌트 간 교차 조회 불가
- 벡터 DB 와 무관한 독립 암호화 계층

기대 효과

- 코사인 유사도 근사 보존 — 의미 기반 검색 가능
- 원본 벡터로부터의 평문 역추론 불가
- GPU 불필요 — 일반 CPU 만으로 실시간 처리
- AES-256 호환 보안강도 유지

95%+

검색 속도 유지율 (원본 대비)

AES-256

호환 보안 강도

동형암호 대비 장점

- GPU / 고성능 연산자원 불필요
- 추론 지연 영향 미미 — 실운영 환경 적용 용이

학습 원장 · 청크 문서 보관 암호화

AES-256 블록 암호화와 RFC 3394 표준 키 관리

문서 원장 암호화

- AES-256 강도의 블록 암호화 적용
- 청크 분할 데이터 및 메타데이터 전 구간 보호
- JSON envelope 구조로 저장
- wrapped_key / nonce / aad / ciphertext 분리 구조
- API 접근 제어를 통과한 호출만 복호화 수행
- RAG-Guard 배포형 패키지로 학습시점 활성화

키 관리 구조 (RFC 3394)

최상위 조직 SEED 키

조직 단위 루트 키 — 오프라인 보관 권장

KEK (Key Encryption Key)

RFC 3394 에 따른 DEK 래핑 전용 키

DEK (Data Encryption Key)

실제 데이터 암호화에 사용되는 키

테넌트별 HKDF 파생키

조직 · 부서 · 테넌트 단위 격리 키 자동 파생

BM25 전문검색 암호화 (HMAC 토큰)

암호화 저장과 전문검색의 양립을 위한 결정적 토큰화 기법

배경 — 일반 암호화는 전문검색과 양립 불가

블록 암호화로 변환된 청크 텍스트는 바이트 단위로 완전 암호화되므로, 형태소 분석기 (OpenSearch nori) 와 BM25 알고리즘이 단어 단위 토큰 매칭을 수행할 수 없습니다. RAG 검색의 첫 단계인 BM25 후보 조회가 불가능해집니다.

해결 방식 — HMAC 결정적 토큰화

원본 평문을 토큰 단위로 HMAC-SHA256 변환하여 OpenSearch 에 저장합니다. 동일 토큰은 동일 HMAC 값을 가지므로 BM25 TF/IDF 통계가 원본과 동일하게 유지되며, 저장 자체는 평문이 아닙니다.

테넌트별 키 파생 (HKDF)

동일 단어라도 조직 A·B 에서 서로 다른 HMAC 토큰 생성. 마스터 키 노출 시에도 조직별 재파생 필요 → 추가 보안 장벽 확보

운영 모드 — STRICT vs 기본

SHIELD_STRICT_MODE 환경변수로 선택. STRICT 모드에서는 text 필드 평문 미저장 → 직접 조회해도 평문 유출 불가

한계 및 대응

와일드카드·퍼지 매칭 불가. 쿼리 전처리 표준화 및 stopword 필터, min-length 조건으로 토큰 빈도 분석 공격 완화

의미 검색 보완

HMAC 단순 일치 한계는 KNN (SRHT 변환 벡터) 계층에서 의미 기반 검색으로 보완

삼중 방어 구조

세 계층을 독립적으로 암호화하여 단일 계층 침해 시에도 평문 노출을 차단

LAYER 01

BM25 전문검색 계층

보호 필드

search_tokens 필드

HMAC-SHA256 으로 결정적 토큰화된 단어 단위
인덱스에서 전문검색 수행 . 평문 노출 없이
BM25 TF/IDF 통계를 그대로 활용합니다 .

LAYER 02

KNN 벡터 검색 계층

보호 필드

embedding 필드

SRXT 변환된 1024 차원 벡터 유사도 검색 수행 .
원본 벡터로부터 평문 역추론이 불가능하면서
의미 기반 검색을 유지합니다 .

LAYER 03

청크 평문 보호 계층

보호 필드

text_encrypted 필드

블록 암호화된 JSON envelope
(wrapped_key, nonce, aad, ciphertext)
구조로 저장 . API 접근 제어 통과 후에만
복호화됩니다 .

단일 계층의 키가 노출되거나 인덱스가 유출되더라도 , 다른 두 계층의 암호가 평문 재구성을 차단합니다 .

API Gateway 구조 및 벡터 DB 호환

국산 API Gateway 및 LLM 연동, CSAP 대응 구조

API Gateway 구성 유연성

- Kong API 기본 적용 / 타 국산 API Gateway 대체 가능
- Kong 기능 (라우팅·TLS 종료·Rate Limiting·CORS) 은 Nginx reverse proxy 대체 가능
- 코드베이스에 Kong 종속성 없음 — HTTP 호출 가능한 어떤 Gateway 와도 연동
- 배포 옵션 : Python 라이브러리 직접 통합 · Docker · TEE 서비스 (Intel TDX, AMD SEV-SNP)

공공 납품 적용 CSAP 기준 국산 API Gateway 적용 및 국산 LLM 연동 가능 — Shield-RAG 는 LLM 에 독립적인 보안 레이어입니다 .

호환 가능한 벡터 DB

구분	DB 종류	지원 인덱스 / 타입	비고
기본 적용	OpenSearch	dense_vector, BM25	ECM·EDR 등 호환 레퍼런스 구현체
호환 가능	Elasticsearch	dense_vector 타입	ESRE 연동
호환 가능	Milvus	IVF_FLAT, HNSW	대규모 벡터 검색
호환 가능	FAISS	IndexFlatIP	임베디드 라이브러리 형태
호환 가능	Pinecone	cosine metric	클라우드 매니지드
호환 가능	Qdrant	cosine distance	Rust 기반 오픈소스

RAG-Guard 적용 매트릭스

데이터 계층별 암호화 적용 및 검색 가능 여부 종합

계층	대상	검색 가능 (벡터 DB)	그래프 DB 적용
프롬프트 / 쿼리 벡터	사용자 질의	○ (유사도)	○
문서 벡터	임베딩 인덱스	○ (유사도)	○ (노드 · 엣지 임베딩)
문서 원장 / 텍스트	원본 + 청크	복호화 필요 (X)	○ (노드 · 엣지 속성)
메타데이터	문서 부가정보	복호화 필요 (X)	○ (관계 메타데이터)
키 관리	DEK / KEK (RFC 3394)	—	○ (동일)
실행 환경	별도 신뢰 환경	—	○ (동일)
전송 구간	네트워크	—	○ (동일)
그래프 구조	토폴로지	—	X (구조 보호 대상 외)

※ 그래프 구조 (토폴로지) 자체는 암호화 대상이 아닙니다 . 노드 · 엣지 연결 관계는 DB 가 순회해야 하므로 구조적으로 보호 불가하며 , 경로 쿼리 접근 패턴은 별도 프라이버시 설계를 권장합니다 .

동형암호 대비 차별점 및 도입 효과

국방 · 공공 환경에서 실운영 가능한 보안 구조

비교 항목	동형암호 방식	RAG-Guard
연산 자원	GPU 및 고성능 연산 필수	일반 CPU 로 처리
추론 지연	10 배 ~ 수백배 지연	원본 대비 95% 이상 성능 유지
전문검색 호환	BM25·형태소 분석 불가	HMAC 토큰으로 BM25 정상 동작
다중 테넌트	별도 설계 필요	조직별 HKDF 파생키 기본 지원
도입 난이도	인프라 · 연산자원 대규모 확충	Python 패키지 / Docker 단순 적용

①

기밀성 확보

원문 · 벡터 · 토큰 전 구간 암호화

②

운영 성능 유지

실시간 검색 · 추론 성능 영향 최소화

③

국산 인프라 적합

국산 Gateway·LLM·DB 연동

④

조직 분리 격리

테넌트 · 부서 단위 키 자동 파생

CONTACT | 감사합니다 .

민감 , 대외비 등 중요정보 환경에 가장 적합한
RAG·LLM 보안 레이어를 제안드립니다 .
