SecureRAG: End-to-End Secure Retrieval-Augmented Generation

Amina Bassit
Michigan State University
`bassitam@msu.edu`

Vishnu Naresh Boddeti
Michigan State University
`vishnu@msu.edu`

Abstract

Retrieval-augmented generation (RAG) enhances large language models (LLMs) with external knowledge from databases but introduces privacy risks when handling sensitive information. Existing defenses fall short: differential privacy degrades accuracy and remains vulnerable to embedding inversion attacks, while fully homomorphic encryption (FHE) ensures security but lacks access control. We present SecureRAG, an end-to-end secure RAG framework that enforces strict access control while mitigating prompt injection data extraction and embedding inversion attacks. It achieves this by decoupling retrieval into secure search and secure document fetching, using FHE for encrypted search and attribute-based encryption (ABE) for fine-grained access control. SecureRAG supports dynamic database updates, adaptive access policies, and integrates seamlessly with FHE-friendly LLMs, adding only 0.05s of overhead. By providing a fully encrypted, privacy-preserving retrieval framework, SecureRAG enables the secure deployment of domain-specific chatbots in sensitive applications.

1 Introduction

Retrieval-augmented generation (RAG)[21] enhances large language models (LLMs)[24] by retrieving relevant information from external sources to generate more accurate and context-aware responses without retraining. By bridging information retrieval and text generation, RAG enables cost-effective chatbot customization across the healthcare, finance, and law sectors. For example, Figure 1 illustrates RAG in healthcare, where a hospital corpus aids doctors (*users*) in generating precise diagnostic suggestions.

Despite its advantages, RAG is highly vulnerable to privacy and security risks in sensitive sectors [36, 30]. The primary threat is *sensitive information leakage*, which can occur through (1) compromised components (e.g., retriever, generator) or (2) adversarial prompt injections that extract restricted data from the database (DB). Beyond direct DB access, exposing cleartext embeddings (queries and documents) risks leaking personally identifiable information (PII). Morris et al. [26] demonstrated that text embeddings are highly invertible, recovering 89% of PII (e.g., full names) from clinical note embeddings, underscoring the need for equal protection of raw text and embeddings. Additionally, Qi et al. [30]

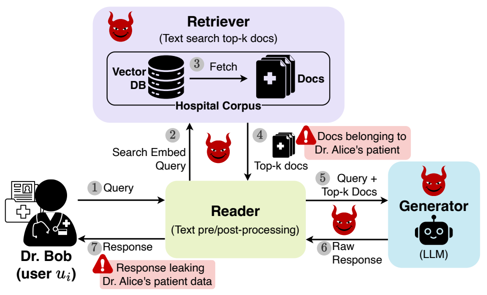


Figure 1: Healthcare-based RAG showing key vulnerabilities: a) retriever exploiting query embeddings and hospital corpus, b) generator misusing queries and retrieved documents, and c) Dr. Bob receiving sensitive data of Dr. Alice's patient due to lack of access control.

introduced a prompt injection data extraction attack that targeted RAG's retrieval DB—rather than the LLM's training data—successfully extracting 41% of a 77K-word book and 3% of a 1.5M-word corpus using only 100 crafted queries. These findings highlight a critical weakness: RAG lacks effective access control at the retrieval level, leaving it vulnerable to data extraction attacks.

These vulnerabilities are evident in the healthcare scenario shown in Figure 1, where Dr. Bob must uphold patient confidentiality while potentially sharing patients with other doctors. He must also be prevented from using prompt injection attacks to access records of Dr. Alice's patients, as they do not collaborate. Deploying separate RAG solutions for each doctor is impractical due to inefficiency, cost, and complexity. Instead, hospitals require a unified RAG system integrated with their database while enforcing strict access controls. The key challenge is ensuring that responses are personalized while *retrieved documents remain both relevant and restricted to authorized patients, preserving security and privacy.*

Most privacy-preserving RAG solutions [16, 4] rely on differential privacy (DP) [11] as a lightweight defense, adding controllable noise to balance privacy and accuracy. However, DP has critical limitations: (1) it distorts text, reducing retrieval and generation accuracy, (2) its privacy guarantee weakens over repeated queries due to budget exhaustion, and (3) it fails to prevent text embedding inversion [19] and prompt injection attacks. These shortcomings make DP unsuitable for an end-to-end, provably secure RAG without accuracy loss.

Fully homomorphic encryption (FHE) [14] has emerged as a promising approach, offering end-to-end security with strong privacy guarantees and no accuracy loss. Despite its runtime overhead, recent advances demonstrated their effectiveness in preventing information leakage from text embeddings [19], enabling secure text classification [2, 20], and supporting secure LLM inference [37, 9, 31]. While FHE provides essential building blocks for a secure RAG, existing solutions do not afford access control on retrieved documents.

We propose SecureRAG, an end-to-end secure RAG framework that enforces access control over retrieved documents while preventing prompt injection data extraction and embedding inversion attacks. SecureRAG achieves secure retrieval by splitting the process into two subphases: (1) secure search and (2) secure document fetching. By leveraging FHE's SIMD property and vertically packing FHE-encrypted embeddings, SecureRAG enables efficient and scalable search over the vector database. To enforce access control, documents are encrypted using an attribute-based encryption (ABE) scheme, ensuring that only authorized users can decrypt retrieved content. SecureRAG supports dynamic management of both the database (e.g., adding/deleting encrypted documents and embeddings) and access rights (e.g., granting/revoking permissions). It seamlessly integrates with FHE-friendly LLM generators [31] without compromising accuracy.

We evaluate SecureRAG to determine whether encryption can be integrated without sacrificing accuracy or hindering seamless model updates. Our results show that SecureRAG matches the accuracy of unprotected RAG across both top-k ranking and context precision metrics, with the latter evaluated by an LLM judge. In terms of performance, SecureRAG retrieves 100 documents from a corpus of 16384 under a 2-attribute policy in just 0.05 seconds on a single GPU—a $13.6\times$ speedup over RemoteRAG [4], which requires two GPUs and 0.68 seconds to retrieve only 5 documents from 160, without any access control and with increased vulnerability to prompt injection attacks.

In summary, SecureRAG seamlessly integrates with FHE-compatible LLMs and supports dynamic access management and database updates, enabling secure and privacy-preserving RAG deployments in sensitive domains such as healthcare.

2 Related Work

In RAG, sensitive data leaks through embedding inversion attacks on its components [26] or prompt injection queries targeting the extraction of restricted documents [30]. Existing solutions protect components separately but remain vulnerable to prompt injection attacks.

Secure text embedding classification. Many solutions mitigate information leakage by encrypting text embeddings with FHE [2, 20, 19] for classification tasks. However, they are limited to basic one-to-one similarity comparisons and do not scale to large databases due to FHE's computational bottlenecks in *search*. SecureRAG addresses this challenge by enabling efficient one-to-many and many-to-many text embedding searches under FHE. It leverages vertical data packing to fully exploit FHE's SIMD property, significantly improving efficiency and reducing encrypted database storage.

Secure inference of LLMs. LLMs are proprietary, requiring queries to be sent to external servers for inference, raising concerns about access and retention of sensitive data. Research on secure inference falls into interactive and non-interactive models. Interactive models [1, 10, 17] rely on multiparty computation (MPC) but suffer from high communication overhead, making them impractical for RAG's large-scale deployment. Hybrid approaches[29] combine HE and MPC to mitigate this overhead. FHE-based solutions [9, 37, 31] offer stronger privacy by transforming LLMs into FHE-friendly architectures, enabling non-interactive, end-to-end encrypted inference. Such architectures focus on redesigning core LLMs' components to better suit encrypted computation. For instance, softmax-based attention mechanisms are replaced with FHE-friendly alternatives such as Gaussian kernel attention, which avoids costly operations such as exponentiation and division that are unsupported by FHE. Also, nonlinear activations (e.g., GELU and ReLU) are approximated using low-degree polynomials to ensure compatibility with FHE. To preserve language capabilities while minimizing computational overhead, lightweight adapters, such as LoRA [18], are used to fine-tune frozen pre-trained LLMs, ensuring that only a small subset of model parameters is subject to FHE constraints. These design techniques shift much of the computation from heavy ciphertext-ciphertext operations to more efficient plaintext-ciphertext operations where possible. SecureRAG is fully compatible with any non-interactive FHE-friendly LLM and can seamlessly switch between different FHE-friendly LLM providers without affecting its retriever or access control policies.

Privacy-preserving RAGs. While individual RAG components can be secured, privacy-preserving RAG solutions aim for end-to-end protection, primarily against information leakage. Existing approaches rely on differential privacy (DP)[11], but its privacy-accuracy tradeoff degrades LLM performance by altering text semantics[23]. DP also fails to prevent embedding inversion attacks[19], and its privacy guarantee weakens with repeated queries, requiring resets that hinder deployment[34]. Moreover, no DP-based RAG solutions prevent prompt injection data extraction attacks. SecureRAG is the first end-to-end secure RAG framework enforcing access control over retrieved documents while preventing information leakage at both the embedding and generator response levels.

3 Preliminaries

SecureRAG integrates encryption with RAG using FHE for computations on encrypted text embeddings and attribute-based encryption to restrict document decryption to authorized users.

3.1 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) [14] enables computations on encrypted data without decryption, providing strong privacy guarantees. Its IND-CPA security [7] prevents semi-honest attackers from inferring plaintexts from ciphertexts. While FHE is computationally expensive, ongoing optimizations, including GPU acceleration [35], have enhanced its practicality. SecureRAG utilizes the CKKS scheme [5] due to its support for floating-point operations, enabling encrypted computations that closely approximate cleartext results. CKKS also leverages the single-instruction multiple-data (SIMD) [33] property, efficiently packing multiple plaintext values into a single ciphertext. However, SIMD efficiency depends on the packing strategy—SecureRAG employs vertical packing, which is optimized for large-scale vector database searches (Figure 2).

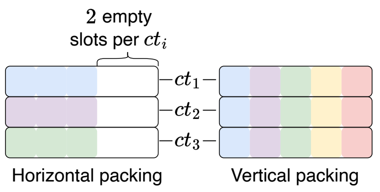


Figure 2: SecureRAG uses the vertical packing to efficiently store the encrypted text embeddings in DB.

3.2 Attribute-Based Encryption

Attribute-Based Encryption (ABE) enables fine-grained access control, allowing decryption only when user attributes meet a defined policy. In Ciphertext-Policy ABE (CP-ABE), access policies are in the ciphertext, giving data owners control, while in Key-Policy ABE (KP-ABE), policies are in decryption keys, managed by a central authority. SecureRAG requires KP-ABE, where hospital authorities control access policies and keys. While most KP-ABE schemes use pairing-based cryptography, LWE and RLWE-based alternatives [8, 22] offer stronger security, and revocability. RLWE-based KP-ABE (Figure 3) enables *key homomorphism*, allowing homomorphic evaluation of public keys over a circuit policy. Selective security (IND-sCPA) [15] ensures adversaries without authorized keys cannot distinguish between encrypted messages. Like FHE, LWE and RLWE-based KP-ABE rely on lattice-based hardness, providing post-quantum security guarantees.

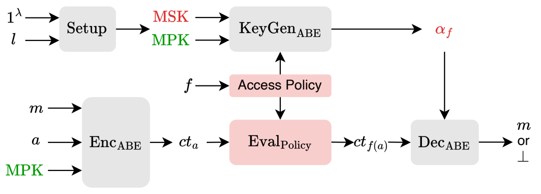


Figure 3: RLWE-based KP-ABE scheme in [8]. More details are in Appendix A.

4 Threat Model

SecureRAG prevents sensitive information leakage across all RAG stages from malicious components (retriever, generator) or users, specifically addressing embedding inversion and prompt injection data extraction attacks. Notably, the latter targets the retrieval database, not the LLM generator's training data, which SecureRAG does not consider in its threat model. We consider a four-party setting with non-colluding entities: users, a reader, a retriever, and a generator. All parties are semi-honest[1], except the reader, a trusted third party representing the hospital administrator. The users, representing hospital staff with attribute-based access, query a customized LLM augmented with the hospital corpus for tailored responses. The reader encrypts sensitive data (doctors' queries, patient records, hospital corpus), sets DB access policies, manages public key infrastructure (PKI), issues user keys, and handles text preprocessing and post-processing. The retriever, a cloud DB provider, stores the encrypted documents and retrieves relevant documents for RAG. The generator, an FHE-friendly proprietary LLM (e.g., OpenAI's ChatGPT), processes encrypted queries and documents, performs inference under encryption, and returns encrypted responses, protecting its intellectual property. The reader protects hospital data while enabling doctors to use external services. The retriever and generator must not extract readable information from queries, corpus, the hospital, or retrieved documents in any form. Users should not intentionally or unintentionally receive responses containing information about other users' sensitive data or documents they are unauthorized to access.

5 SecureRAG

In RAG, the retriever and generator use different text embedding models suitable for text similarity or generation. Thus, SecureRAG extracts the generator's text embeddings from raw documents on the fly to support any FHE-compatible LLM generator. This enables SecureRAG to integrate with any FHE-friendly LLM generator.

5.1 Key Generation

SecureRAG encrypts text embeddings with FHE, using separate keypairs for the retriever and generator, and encrypts raw documents with KP-ABE to enforce access policies. During setup, the reader, as the system administrator, generates two FHE keypairs: (pk_r, sk_r) for retriever searches and (pk_g, sk_g) for generator inference. It shares pk_r with the retriever and pk_g with the generator, keeping sk_r and sk_g private. The reader also generates a KP-ABE master keypair $(\texttt{MPK}, \texttt{MSK})$, sharing $\texttt{MPK}$ with users while keeping $\texttt{MSK}$ private.

5.2 Encrypted Vector and Document DBs

To build an end-to-end secure RAG while maintaining high efficiency, SecureRAG splits the encrypted vector database (DB) into two parts: an FHE-encrypted chunked vector DB and an ABE-encrypted raw document DB enforcing an access control policy over the retrieved documents.

FHE-Encrypted chunked vector DB. The reader encrypts the vector DB following its chunking strategy that optimizes the overall RAG performance. SecureRAG enables the reader to pack n

[1]Follow the protocol but try to infer sensitive information.

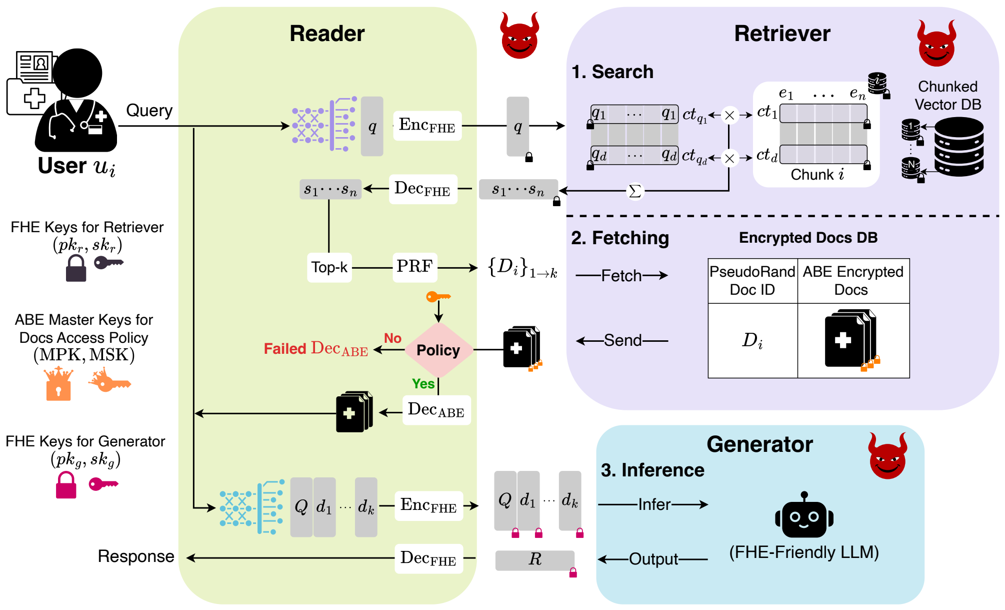

Reader
Retriever
1. Search
Chunked Vector DB
Chunk i
2. Fetching
Encrypted Docs DB
PseudoRand Doc ID
ABE Encrypted Docs
Generator
3. Inference
(FHE-Friendly LLM)
User u_i
Query
Response
FHE Keys for Retriever (pk_r, sk_r)
ABE Master Keys for Docs Access Policy (MPK, MSK)
FHE Keys for Generator (pk_g, sk_g)
$\mathrm{Enc}_{\mathrm{FHE}}$
$\mathrm{Dec}_{\mathrm{FHE}}$
Top-k
PRF
$\{D_i\}_{1\to k}$
Fetch
Send
Policy
No
Yes
Failed $\mathrm{Dec}_{\mathrm{ABE}}$
$\mathrm{Dec}_{\mathrm{ABE}}$
Infer
Output
R

Figure 4: Overview of the SecureRAG framework integrating an FHE-friendly LLM generator with ABE-encrypted documents, ensuring fine-grained access control. 1) The user's query embedding is FHE-encrypted and used to search for the top-k document indexes in the encrypted chunked vector DB. 2) The ABE-encrypted documents are fetched via pseudo-random IDs and only decryptable if the user u_i satisfies the access policy. Finally, 3) the embeddings of the query and top-k documents are FHE-encrypted and sent to the LLM generator for secure inference, generating an encrypted response, which is post-processed by the reader before delivering it to the user.

embeddings $E = \{e_i\}_{i\in[1,n]}$ of dimension d using only d ciphertexts, where $e_i = (e_{1,i}, \cdots, e_{d,i})$ and n is ciphertext capacity and $n >> d$, which is also the chunk's size. This is achieved by vertically arranging each chunk's embeddings and packing them row-wise with one ciphertext $ct_j = \mathrm{Enc}_{\mathrm{FHE}}^{pk_r}(e_{j,1}, \cdots, e_{j,n})$ per row, resulting in d ciphertexts per chunk $ct_E = \{ct_j\}_{j\in[1,d]}$.

ABE-Encrypted document DB. The raw documents $\{\mathtt{Doc}_t\}_t$ belonging to user u_i are first ABE-encrypted under user's attributes $a_{u_i} \in \{0,1\}^l$ using the master public key $\mathtt{MPK}$ to yeild the ciphertext $ct_{D_t} = \mathrm{Enc}_{\mathrm{ABE}}^{\mathtt{MPK}, a_{u_i}}(\mathtt{Doc}_t)$ over which the reader evaluates the circuit policy f, resulting in $ct^f_{D_t} = \mathrm{Enc}_{\mathrm{ABE}}^{\mathtt{MPK}, f(a_{u_i})}(\mathtt{Doc}_t)$, ciphertexts decryptable with the policy secret key α_f if and only if a_{u_i} satisfies the policy f. To enable a secure direct fetch of those encrypted documents and avoid storing their respective IDs, we use a keyed pseudo-random function [3] $D_t = \mathrm{PRF}_K(d_t)$ that given a secret key K and an index d_t it returns the same pseudorandom D_t completely different from d_t. The reader generates and sends the ABE-encrypted documents $\{ct^f_{D_t}\}_t$ along with their pseudorandom identifiers $\{D_t\}_{t\in[1,k]}$ to the retriever for storage. SecureRAG benefits from this by making fetching secure documents efficient without the risk of leaking their actual identifiers, which saves storage.

5.3 Protocol Description

SecureRAG, depicted in Figure 4, consists of three steps: 1) FHE-encrypted search for the top-k document indexes, 2) secure fetching of the ABE-encrypted documents via pseudorandom identifiers, and 3) FHE-encrypted LLM inference. To enhance efficiency, the retrieval part is split into 1) and 2), with SecureRAG filtering documents by user attributes and access policy before step 3).

Searching top-k document indexes. We consider the query embedding q and document embeddings $E = \{e_i\}_{i\in[1,n]}$ as normalized d-dim vectors. Normalized vectors enable efficient encrypted search, as the inner product (IP) is cheaper to compute under encryption than cosine similarity while preserving identical scores. The equation below shows that the inner product avoids computing the embedding norms, which would require expensive computation under FHE.

$$\text{Cosine}(\tilde{q}, \tilde{e_j}) = \frac{\langle \tilde{q}, \tilde{e_j} \rangle}{\|\tilde{q}\| \cdot \|\tilde{e_j}\|} = \langle q, e_j \rangle$$

A naive encrypted search over n embeddings would compare the query against one embedding at a time, resulting in n IPs costing n homomorphic multiplications. By leveraging the SIMD property and the vertical packing of the document vector DB, SecureRAG computes those n IPs at once, reducing the computation cost to only d homomorphic multiplications, where $d << n$, while the search remains exhaustive. Hence, the reader extracts the user query embedding $q = (q_1, \cdots, q_j, \cdots, q_d)$ that is compatible with the retriever's embedding model. Then, it encrypts each component q_j as an n-dim vector of its replica, yeilding $ct_{q_j} = \text{Enc}^{pk_r}_{\text{FHE}}(q_j, \cdots, q_j)$. Next, it sends $ct_q = \{ct_{q_j}\}_{j \in [1,d]}$ to the retriever for computing IP w.r.t. each chunk as follow:

$$\langle ct_q, ct_E \rangle = \sum_{j \in [1,d]} q_j \times ct_j$$

The retriever returns $ct_S = \text{Enc}^{pk_r}_{\text{FHE}}(s_1, \cdots, s_n)$ a ciphertext containing the IP scores $s_j = \langle q, e_j \rangle$. The reader decrypts the scores $\text{Dec}^{sk_r}_{\text{FHE}}(ct_S) = (s_1, \cdots, s_n)$, sorts them, and selects the top-k.

Fetching relevant documents. From the indexes $\{\mathrm{d}_t\}_{t \in [1,k]}$ of the top-k scores, the reader recovers the documents' pseudorandom identifiers D_t. Next, the reader sends $\{D_t\}_{t \in [1,k]}$ to the retriever who sends back the ABE-encrypted documents $\{ct^f_{D_t}\}_{t \in [1,k]}$ where $ct^f_{D_t} = \text{Enc}^{\texttt{MPK}, f(a_{u_i})}_{\text{ABE}}(\texttt{Doc}_t)$ with user's u_i attributes a_{u_i}, an l-dim binary vector with l being the maximum number of attributes a user can have. Then, the reader decrypts documents using the secret key α_f for access policy f. Decryption succeeds only if user u_i's attributes a_{u_i} satisfy f; otherwise, it fails.

Encrypted inference. The reader combines the user's query with the successfully ABE-decrypted documents and extracts their embeddings with an embedding model compatible with the generator. It then sends their FHE-encryption using pk_g to the FHE-friendly LLM generator, performs the secure inference under encryption, and returns its encrypted response back to the reader. Subsequently, the reader decrypts it using sk_g, post-processes it, and displays it to the user.

5.4 Complexity and Security Analyses

Table 1 presents SecureRAG's computational complexity and storage requirements for its secure search part, which is the dominant part of the retrieval. Note that FHE schemes can adjust their parameters to expand ciphertext capacity as needed, which would help in improving efficiency. Our security analysis is in Appendix B.

Table 1: SecureRAG's retriever search storage and computational complexity as $\mathcal{O}(N \cdot (\#\text{Add}_{\text{HE}} + \#\text{Mult}_{\text{HE}}))$ where N is the number of chunks.

Dimension		d	256	768
Complexity	Add_{HE}	$d-1$	255	767
	Mult_{HE}*	d	256	768
Storage	Enc Query	d	256	768
	Enc DB	$d \cdot N$	$256 \cdot N$	$768 \cdot N$

* Those are homomorphic multiplications of depth 1.

5.5 Dynamic Databases and Access Rights

SecureRAG efficiently and dynamically handles the addition and deletion of documents and their embeddings with on-the-fly user management, including dynamic addition, revocation, and real-time policy updates.

Dynamic Databases. For the **addition** of a document $\texttt{Doc}_t$, the reader updates the vector DB with the document embedding $\bar{e}_t = (\bar{e}_{1,t}, \cdots, \bar{e}_{d,t})$ by selecting a chunk with an available empty slot d_t, and sends to the retriever the set of ciphertexts encrypting $\bar{e}_t$, that is, $ct_{\bar{e}_t} = \{ct_{\bar{e}_{j,t}}\}_{j \in [1,d]}$ where $ct_{\bar{e}_{j,t}} = \text{Enc}^{pk_r}_{\text{FHE}}(\cdots, 0, \bar{e}_{j,t}, 0, \cdots)$. Then, the retriever then updates the encrypted vector DB $ct_E = \{ct_j\}_{j \in [1,d]}$ using one homomorphic addition per ciphertext $ct_j := ct_j + ct_{\bar{e}_{j,t}} \forall j \in [1, d]$. Also, the reader generates the document's pseudorandom identifier $D_t = \text{PRF}_K(d_t)$ and ABE-encryption of the document to which it applies the access policy f resulting in $ct^f_{D_t} = \text{Enc}^{\texttt{MPK}, f(a_{u_i})}_{\text{ABE}}(\texttt{Doc}_t)$. It sends to the retriever $ct^f_{D_t}$ for storing it under D_t. Note that our addition can add batch of embeddings at once $\bar{e}_{t_1}, \cdots, \bar{e}_{t_m}$ at once with $ct_{\bar{e}_{j,t}} = \mathrm{Enc}_{\mathrm{FHE}}^{pk_r}(\cdots, 0, \bar{e}_{j,t_1}, \cdots, \bar{e}_{j,t_m}, 0, \cdots)$. SecureRAG supports batch **deletion** of embeddings. The reader creates a deletion vector $v \in \{0, -1\}^n$, marking deletions with -1, and encrypts it as $ct_v = \mathrm{Enc}_{\mathrm{FHE}}^{pk_r}(v)$. The retriever updates the encrypted DB $ct_E = \{ct_j\}_{j \in [1,d]}$ using one homomorphic addition and multiplication per ciphertext: $ct_j := ct_j + ct_v \times ct_j \quad \forall j \in [1, d]$. The retriever processes encrypted updates blindly, without knowing the modified documents.

Dynamic access control. The KP-ABE-SW scheme [22] supports switchable attributes, enabling dynamic user management and policy updates. SecureRAG leverages these capabilities to manage access rights dynamically, with all modifications handled by the reader, acting as the system administrator.

6 Experiments

Our goal is not to enhance state-of-the-art RAG accuracy but to assess whether an encryption layer can be integrated without compromising accuracy or efficiency in sensitive applications. The experiments in Section 6.1 were conducted using Python 3.12 on a NVIDIA RTX A6000 GPU-equipped server. For embedding search, we implemented CUDA C++ experiments using PhantomFHE[35] with the CKKS scheme, tested on HPCC with a single NVIDIA A100 core and 16GB of memory. For document decryption, we used PALISADE-abe[28], implementing the lattice-based ABE scheme in C++, running on an Apple M3 Pro (12 cores, 36GB RAM). We will publicly release our code[2].

6.1 Performance Evaluation

We evaluate SecureRAG using the standard RAG assessment pipeline [12, 32], where an LLM judge systematically assesses retriever and generator performance. The judge receives instructions, the query, retrieved documents (retriever output), and the generated response (LLM output) for structured analysis. To evaluate the impact of encryption on RAG, we apply rounding to 5, reflecting CKKS's precision limits. Performance in cleartext (no rounding) is compared to the encrypted setting (with rounding) to measure potential losses.

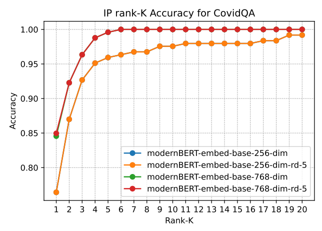
Models. We use ModernBERT Embed [27] for retrieval (supporting 256 and 768 dimensions) and Llama-2-7B [25] as the FHE-friendly generator, optimized for GPUs [6, 31]. SecureRAG is tested with ModernBERT Embed as the retriever and Llama-2-7B as the generator, using Llama-3.1-8B as the LLM judge, without fine-tuning any models.

Datasets. To assess the adaptation of RAG to domain-specific contexts, we consider the following datasets provided in [13]: the PubMedQA and CovidAQ-RAG datasets for the biomedical domain, the TechQA dataset for customer support, and the FinQA and TAT-QA datasets for the financial domain.

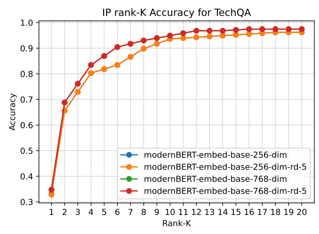
Metrics. We measure the retriever's effectiveness using the *rank top-k* and the *context precision* metric as defined in [12], providing the LLM judge with a specific prompt (see Appendix D) instructing it to return a verdict based on the relevance of the retrieved documents w.r.t. the question and the LLM generator's response. High scores indicate high performance. We note that separate answer-only evaluations are not repeated here, as the answer quality of FHE-compatible LLMs has already been rigorously validated in prior work [31], showing negligible differences from non-FHE LLMs. Since SecureRAG integrates these models without modification, it inherits their accuracy performance.

Assessment. For each dataset, the `documents` column contains documents corresponding to each `question`. We combined all documents to build the vector DB, extracted their normalized embeddings for $d \in \{256, 768\}$, and rounded them to 5. Figure 5 shows that rounding has no impact on retrieval accuracy, as ranking curves with and without rounding overlap almost perfectly, regardless of the embedding dimensionality. Table 2 reports LLM judged context precision for varying retrieved documents ($k \in \{1, 5, 10\}$). The LLM judge produces nearly identical scores for $k \geq 5$, with minor variations at $k = 1$, which can be due to its probabilistic nature. SecureRAG maintains RAG accuracy with minimal loss when retrieving one document while preventing prompt injection and data extraction attacks. See Appendix C for PubMedQA and TAT-QA results.

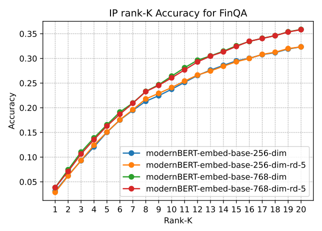
[2]Upon acceptance at `https://github.com/aminabassit/secureRAG`



(a) CovidQA



(b) TechQA



(c) FinQA

Figure 5: SecureRAG's retriever performance on CovidQA, TechQA, and FinQA (rounded to 5, orange and red curves) meets unprotected RAG accuracy for rank top-k with $k \in [1, 20]$ for embeddings of dimensions $d \in \{256, 768\}$.

Table 2: Performance of SecureRAG for context precision using a retriever with d-dim embeddings. Gray (resp. white) cells are with (resp. without) rounding.

Domain	d-dim	256			768		
	K Docs	1	5	10	1	5	10
Healthcare	PubMedQA	0.889	0.999	0.999	0.839	0.999	0.999
		0.851	0.999	0.999	0.847	0.999	0.999
	CovidQA	0.974	0.999	0.999	0.961	0.999	0.999
		0.971	0.999	0.999	0.994	0.999	0.999
Finance	TAT-QA	0.929	0.999	0.929	0.911	0.999	0.999
		0.938	0.999	0.998	0.916	0.999	0.999
	FinQA	0.955	0.999	0.999	0.970	0.999	0.999
		0.973	0.999	0.999	0.961	0.999	0.999
Tech	TechQA	0.999	0.999	0.999	0.999	0.999	0.999
		0.999	0.999	0.999	0.999	0.999	0.999

6.2 Runtime Evaluation

We measure SecureRAG's latency for its 3 steps: (1) FHE-encrypted search on GPU, (2) ABE-based secure fetching on CPU, and (3) FHE-encrypted LLM inference, referencing reported GPU benchmarks for state-of-the-art FHE-friendly LLMs.

Search Performance. Table 3 reports SecureRAG's mean runtime for top-100 document searches using embeddings of dimensions $d \in \{256, 512, 768, 1024\}$, evaluated at two chunk capacities (16384 and 32768 embeddings per chunk). For single-chunk searches, runtime ranges from 18ms to 47ms at 16384 capacity and 30ms to 71ms at 32768, showing efficiency gains with larger chunks. For large-scale DBs, SecureRAG scales effectively: retrieving from 1M embeddings requires 30 chunks, adding < 2.2s in runtime, while retrieval from 1B embeddings spans $\sim$ 30K chunks in 36.11 minutes. SecureRAG achieves a $13.6\times$ speedup over RemoteRAG [4], which takes 0.68s on two GPUs to retrieve just 5 documents from 160, lacking access control and remaining vulnerable to prompt injection data extraction attacks.

Access Control Overhead. Table 4 measures ABE-decryption runtime for 1 to 100 encrypted documents ($\sim$ 350–700 words) with 2 to 10 attributes. Decryption time scales with attributes but remains minimal compared to search, peaking at 92.43ms, demonstrating efficient enforcement of access control with low overhead.

Table 3: SecureRAG's GPU mean runtime for searching the top-100 embeddings chunk-wise, with N embeddings per chunk at a 128-bit security level.

Dimension	256	512	768	1024	#Chunks
N	16384	16384	16384	16384	–
100 : N	18ms	27ms	37ms	47ms	1
100 : 10^6	1.11s	1.67s	2.29s	2.91s	61
100 : 10^9	18.31min	27.46min	37.63min	47.81min	61035
N	32768	32768	32768	32768	–
100 : N	30ms	43ms	58ms	71ms	1
100 : 10^6	0.93s	1.33s	1.79s	2.2s	30
100 : 10^9	15.25min	21.87min	29.5min	36.11min	30517

* Runtime measured over 500 iterations for 100 : N and extrapolated for 100 : 10^6 and 100 : 10^9.

Table 4: SecureRAG's CPU mean runtime for ABE-decrypting K documents assuming one document per ciphertext using a 128-bit security level.

# Attributes	2	4	6	8	10
16384 bits documents $\sim$ 350 words					
1 doc	0.47ms	1.04ms	1.23ms	2.03ms	2.25ms
10 docs	1.20ms	2.14ms	3.15ms	4.09ms	5.08ms
100 docs	13.81ms	20.53ms	28.04ms	35.29ms	41.44ms
32768 bits documents $\sim$ 700 words					
1 doc	1.45ms	3.03ms	4.08ms	5.12ms	6.08ms
10 docs	3.13ms	5.30ms	7.37ms	9.31ms	11.20ms
100 docs	27.69ms	45.49ms	61.55ms	76.62ms	92.43ms

* Runtime measured over 500 iterations.

End-to-End Runtime. Table 5 compares SecureRAG with existing approaches that protect only specific RAG components. SecureRAG adds just 0.05s overhead to FHE-friendly non-interactive LLMs, leading to a total runtime of 26.55s to 37.35s. Thus, SecureRAG effectively mitigates prompt injection data extraction attacks while seamlessly integrating with state-of-the-art FHE-friendly LLMs, ensuring strong privacy protection at minimal computational cost.

Table 5: Runtime comparison between SecureRAG and state-of-the-art solutions showing a seamless integration of SecureRAG with the state-of-the-art FHE-friendly LLMs, incurring negligible overhead while effectively preventing prompt injection data extraction attacks.

Solution	Approach	Docs	DB	Dim	Tokens	GPUs	Retriever (s)	Generator (s)	Total (s)	PIDE[*] Attack	Access Control
[19]	FHE	1	1000	768	-	-	0.6	-	-	**Vulnerable**	✗
RemoteRAG [4]	DP&PHE	5	160	768	-	2	0.68	-	-	**Vulnerable**	✗
BOLT [29]	HE&MPC	-	-	768	128	4	-	185	-	**Vulnerable**	✗
NEXUS [37]	FHE	-	-	768	128	4	-	37.3	-	**Vulnerable**	✗
HEaaN [31]	FHE	-	-	768	128	1	-	26.5	-	**Vulnerable**	✗
SecureRAG+NEXUS†	FHE&ABE	100	16384	768	128	4	0.05	37.30	37.35	**Prevented**	✓
SecureRAG+HEaaN†	FHE&ABE	100	16384	768	128	1	0.05	26.50	26.55	**Prevented**	✓

[*] Prompt injection data extraction (PIDE) attack [30]. † Retrieval of 100 documents from a 16384 vector DB using 2 attributes 51.81ms.

7 Limitations

SecureRAG has the following limitations. Its overall runtime is heavily influenced by the efficiency of the FHE-friendly LLM generator it integrates with. Additionally, it operates in three rounds, two of which involve the retriever; reducing these interactions could improve efficiency. Another limitation is that the number of supported attributes is fixed during setup, requiring careful estimation of expected attributes. Increasing this number would slightly impact the retriever's runtime. Another limitation of SecureRAG is the heavy key management burden on the reader's side. As a trusted third party system administrator, the reader is responsible for handling cryptographic keys for ABE and FHE operations, including key distribution, updates, and revocations. This overhead can increase storage complexity and require efficient key management strategies to maintain scalability. Also, a trusted third party introduces a single point of failure, as if it is compromised, could undermine the entire system; however, in practice, many real-world deployments rely on a centralized authority for efficiency and trust management. This risk can be mitigated through decentralization schemes such as multi-authority ABE (MA-ABE) or threshold cryptography to ensure no single entity has absolute control.

8 Conclusion

Privacy and security are critical for responsible RAG deployment, especially in healthcare, where unauthorized access can lead to severe violations. This paper presents SecureRAG, an end-to-end secure RAG framework that integrates FHE and ABE to prevent information leakage, enforce access control, and defend against prompt injection and embedding inversion attacks. SecureRAG splits retrieval into secure search and secure document fetching, ensuring only authorized users access relevant documents without compromising accuracy. Our evaluation shows SecureRAG matches unprotected RAG in rank top-k and context precision metrics. With a single GPU, it retrieves 100 documents from 16K under a 2-attribute policy in 51.81ms, achieving a 13× speedup over existing solutions, which retrieve only 5 documents from 160 with no access control. SecureRAG supports dynamic database updates and adaptive access control while seamlessly integrating with FHE-friendly LLMs, adding only 0.05s of overhead. By effectively preventing prompt injection data extraction attacks, SecureRAG provides a scalable, practical solution for privacy-preserving RAG deployments. It addresses core security challenges, laying the foundation for future research on secure and privacy-aware chatbots.

References

[1] Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. Privformer: Privacy-preserving transformer with mpc. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 392–410. IEEE, 2023. `https://ieeexplore.ieee.org/document/10190506`.

[2] Ahmad Al Badawi, Louie Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. Privft: Private and fast text classification with homomorphic encryption. *IEEE Access*, 2020. https://ieeexplore.ieee.org/abstract/document/9296754.

[3] Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In *Annual Cryptology Conference*. Springer, 2010.

[4] Yihang Cheng, Lan Zhang, Junyang Wang, Mu Yuan, and Yunhao Yao. Remoterag: A privacy-preserving llm cloud rag service. *arXiv preprint arXiv:2412.12775*, 2024.

[5] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*. Springer, 2017. https://link.springer.com/chapter/10.1007/978-3-319-70694-8_15.

[6] CKKS Community, 2024. https://iheaan.com/?target=heaan-llm.

[7] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European transactions on Telecommunications*, 1997.

[8] Wei Dai, Yarkın Doröz, Yuriy Polyakov, Kurt Rohloff, Hadi Sajjadpour, Erkay Savaş, and Berk Sunar. Implementation and evaluation of a lattice-based key-policy abe scheme. *IEEE Transactions on Information Forensics and Security*, 13(5):1169–1184, 2017.

[9] Leo de Castro, Antigoni Polychroniadou, and Daniel Escudero. Privacy-preserving large language model inference via gpu-accelerated fully homomorphic encryption. In *Neurips Safe Generative AI Workshop 2024*, 2024.

[10] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Chen. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533*, 2023. https://arxiv.org/pdf/2307.12533.

[11] Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*. Springer, 2006.

[12] Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation. *arXiv preprint arXiv:2309.15217*, 2023.

[13] Robert Friel, Masha Belyi, and Atindriyo Sanyal. Ragbench: Explainable benchmark for retrieval-augmented generation systems. *arXiv preprint arXiv:2407.11005*, 2024.

[14] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009.

[15] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, 2006.

[16] Nicolas Grislain. Rag with differential privacy. *arXiv preprint arXiv:2412.19291*, 2024.

[17] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure GPT inference with function secret sharing. *Cryptology ePrint Archive*, 2023. https://eprint.iacr.org/2023/1269.pdf.

[18] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 2022. https://arxiv.org/pdf/2106.09685v1/1000.

[19] Donggyu Kim, Garam Lee, and Sungwoo Oh. Toward privacy-preserving text embedding similarity with homomorphic encryption. In *Proceedings of the Fourth Workshop on Financial Technology and Natural Language Processing (FinNLP)*, 2022. https://aclanthology.org/2022.finnlp-1.4.pdf.

[20] Garam Lee, Minsoo Kim, Jai Hyun Park, Seung-won Hwang, and Jung Hee Cheon. Privacy-Preserving Text Classification on BERT Embeddings with Homomorphic Encryption. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2022. `https://aclanthology.org/2022.naacl-main.231.pdf`.

[21] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 2020.

[22] Fucai Luo, Haiyan Wang, Xingfu Yan, and Jiahui Wu. Key-policy attribute-based encryption with switchable attributes for fine-grained access control of encrypted data. *IEEE Transactions on Information Forensics and Security*, 2024.

[23] Justus Mattern, Benjamin Weggenmann, and Florian Kerschbaum. The limits of word level differential privacy. In *Findings of the Association for Computational Linguistics: NAACL 2022*, 2022.

[24] Xiangbin Meng, Xiangyu Yan, Kuo Zhang, Da Liu, Xiaojuan Cui, Yaodong Yang, Muhan Zhang, Chunxia Cao, Jingjia Wang, Xuliang Wang, et al. The application of large language models in medicine: A scoping review. *Iscience*, 27(5), 2024.

[25] Meta. Meta llama 2-7b, 2024. `https://huggingface.co/meta-llama/Llama-2-7b`.

[26] John Xavier Morris, Volodymyr Kuleshov, Vitaly Shmatikov, and Alexander M Rush. Text embeddings reveal (almost) as much as text. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. `https://openreview.net/pdf?id=EDuKP7DqCk`.

[27] Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. Nomic embed: Training a reproducible long context text embedder, 2024.

[28] PALISADE. Palisade abe - lattice cryptography library experimental repository, 2020. `https://gitlab.com/palisade/palisade-abe`.

[29] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers. In *Symposium on Security and Privacy (SP)*. IEEE, 2024. `https://ieeexplore.ieee.org/abstract/document/10646705`.

[30] Zhenting Qi, Hanlin Zhang, Eric Xing, Sham Kakade, and Himabindu Lakkaraju. Follow my instruction and spill the beans: Scalable data extraction from retrieval-augmented generation systems. *arXiv preprint arXiv:2402.17840*, 2024.

[31] Donghwan Rho, Taeseong Kim, Minje Park, Jung Woo Kim, Hyunsik Chae, Jung Hee Cheon, and Ernest K Ryu. Encryption-friendly llm architecture. *arXiv preprint arXiv:2410.02486*, 2024. `https://arxiv.org/pdf/2410.02486`.

[32] Aymeric Roucher. RAG Evaluation - Hugging Face Open-Source AI Cookbook, 2024. `https://huggingface.co/learn/cookbook/en/rag_evaluation`.

[33] Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 2014.

[34] Lauren Watson. An introduction to differential privacy, 2020. `https://laurenwatson.github.io/blogposts/2020-10-25-dp/`.

[35] Hao Yang, Shiyu Shen, Wangchen Dai, Lu Zhou, Zhe Liu, and Yunlei Zhao. Phantom: A cuda-accelerated word-wise homomorphic encryption library. *IEEE Transactions on Dependable and Secure Computing*, 2024. doi: 10.1109/TDSC.2024.3363900. `https://ieeexplore.ieee.org/abstract/document/10428046`.

[36] Shenglai Zeng, Jiankun Zhang, Pengfei He, Yue Xing, Yiding Liu, Han Xu, Jie Ren, Shuaiqiang Wang, Dawei Yin, Yi Chang, et al. The good and the bad: Exploring privacy issues in retrieval-augmented generation (rag). *arXiv preprint arXiv:2402.16893*, 2024.

[37] Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. *Cryptology ePrint Archive*, 2024. `https://eprint.iacr.org/2024/136.pdf`.

A RLWE-based KP-ABE Scheme

We recall the RLWE-based KP-ABE scheme proposed in [8] and enhanced with attribute revocability in [22].

- $\texttt{Setup}(1^\lambda, l) \rightarrow \{\texttt{MPK}, \texttt{MSK}\}$ where λ is the security parameter, l is the number of user maximum attributes, MPK is public master key, and MSK is secret master key.
- $\texttt{Enc}_{\texttt{ABE}}(m, a, \texttt{MPK}) \rightarrow ct_a$ where m is the message, a the user's attributes, and ct_a the ouptut ciphertext linked to the user's attributes.
- $\texttt{KeyGen}_{\texttt{ABE}}(\texttt{MSK}, \texttt{MPK}, f) \rightarrow \alpha_f$ where f is the circuit policy, which is a boolean circuit, and α_f is the policy decryption key.
- $\texttt{Eval}_{\texttt{ABE}}(ct_a, f) \rightarrow ct_{f(a)}$ where $ct_{f(a)}$ is a ciphertext linked to the policy f.
- $\texttt{Dec}_{\texttt{ABE}}(ct_{f(a)}, \alpha_f, \tilde{a}) \rightarrow \bar{m} \texttt{ or } \perp$ where $\bar{m}$ the recovered message if the attribute $\tilde{a}$ satisfies the policy f otherwise the decryption fails $\perp$.

B SecureRAG Security Analysis

Our security analysis follows our threat model discussed in Section 4, where the parties are assumed semi-honest and non-colluding, except for the reader, who is a trusted third party. We recall that semi-honest parties adhere to the protocol as specified but aim to infer sensitive information about other participants solely through their interactions.

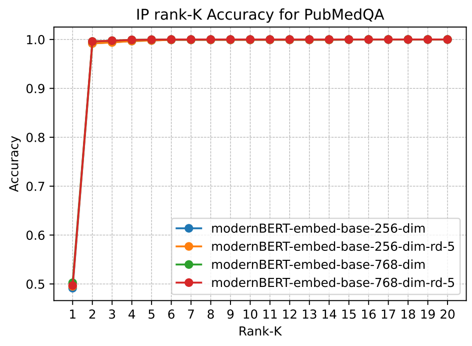
Compromised User. A semi-honest user, who interacts with the system only by sending queries and receiving responses, may attempt to extract information about other users' documents. They could do this by crafting malicious queries targeting unauthorized documents in the database. However, SecureRAG prevents such attacks through ABE-encryption of documents. Even if a query matches an unauthorized document, decryption will fail because the document's ciphertext is bound to a policy f that excludes the user's attributes.

Compromised Retriever. A semi-honest retriever that stores the ABE-encrypted documents and their FHE-encrypted embeddings can try to learn the user query, which is FHE-encrypted, the retrieved documents under both forms. For the FHE-encrypted query and embeddings, a compromised retriever cannot infer meaningful information thanks to the IND-CPA security property of FHE, which ensures that FHE ciphertexts remain indistinguishable, even when their underlying plaintexts are identical. The ABE-encrypted documents are fetched using pseudorandom identifiers that look like random values to the retriever. Thus, it cannot learn which documents are fetched. Moreover, the RLWE-based KP ABE scheme that encrypts the documents satisfies the *selective security (IND-sCPA)* property [15] that prevents an attacker, who claimed to possess certain attributes from between ciphertexts of two chosen plaintexts as long as they do not satisfy the access policy f.

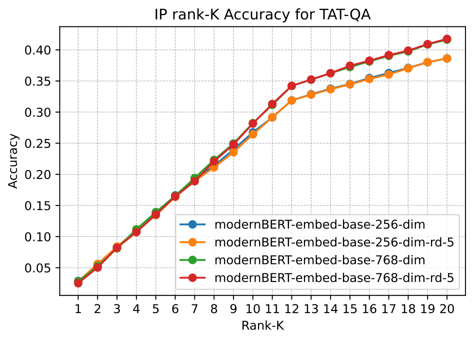
Compromised Generator. A semi-honest generator that receives encrypted and top-k documents and returns its response encrypted can try to infer information about the query and the selected documents. However, this is prevented by the IND-CPA property of FHE, which entails that the FHE ciphertexts cannot be distinguished even if their underlying plaintexts are identical. Given that, such a compromised generator performs the inference on encrypted data protected by the IND-CPA property, it will be incapable of learning any meaningful information.

C SecureRAG performance on other datasets

Similarly to Figure 5, Figure 6 shows that rounding does not affect retrieval accuracy, as the ranking curves for the PubMedQA and TAT-QA datasets with and without rounding align almost perfectly, irrespective of embedding dimensionality.



(a) PubMedQA



(b) TAT-QA

Figure 6: SecureRAG's retriever performance on the PubMedQA and TAT-QA datasets (rounded to 5, orange&red curves) matches unprotected RAG accuracy for rank top-k with $k \in [1, 20]$.

D Prompts for the LLM judge

We used the following prompt from RAGAS [12] for our evaluation of the context precision metric.

Context Precision Prompt

- Instruction: Given question, answer and context verify if the context was useful in arriving at the given answer. Give verdict as 1 if useful and 0 if not.
- Prompt: '`instruction` question: `question` context: `context` answer: `answer` verdict: '