xxx: Secure and Efficient Similarity Search in SGX

Paper ID:

1 Background and Problem

1.1 Similarity Search

With the development of the network, the demand for multimedia data similarity search by users is rapidly increasing. Nowadays, the main method of realizing multimedia data similarity search is comparing the similarity of features which is extracted from original data. The process of extracting features from data ensures the similarity relationship by using hashing [10, 17, 20]. For two multimedia data, if their features are similar, then they are also similar. For simple calculation and efficient storage, the extracted features are represented as binary vector [12, 17, 18]. The define of similarity is the hamming distance between two binary vectors of features is no more than threshold predetermined. The smaller the hamming distance is, the more similar two features are.

Multi Index Hashing. A direct way of similarity search is to perform linear scanning, which is not feasible for large datasets. Another way is to create a hash table, but when the length of the feature is too long and the Hamming distance threshold is too large, the search efficiency will rapidly decrease. *Multi Index Hashing* is the main method of similarity search, which is usefully reduces search space for similar queries but at the cost of increasing memory usage [11, 13]. The structure of Multi Index divide the feature with length of n into m disjoint sub feature segments.Each segment has $\lfloor n/m \rfloor$ or $\lceil n/m \rceil$ bits and stores separately in m index called sub index. What the sub index stores is the KV pair {sub feature, identifier}, in which the identifier can uniquely identify a feature. Assuming that the Hamming distance threshold for two multimedia data with similar feature values is k, according to the pigeonhole principle [13], the sub humming distance for each segment is $\lfloor k/m \rfloor$. As shown in the Figure1, in searching process of query q, it firstly finds all sub feature that hamming distance between it and q is no more than $\lfloor k/m \rfloor$ separately in each sub index and puts the corresponding identifiers into candidate pool. Secondly, it compares q with the feature in full index that stores KV pair {identifiers, feature } according to the identifiers in candidate pool, if the hamming distance between the feature and q is less than k, then the feature is similar with q. Compared with using , using normal hash index, Multi Index let one query' search times decreases from $L(n, k)$ to $m*L(n/m, k/m)$ [13], where

$$L(n,k) = \sum_{i=0}^{k} \binom{n}{i}$$

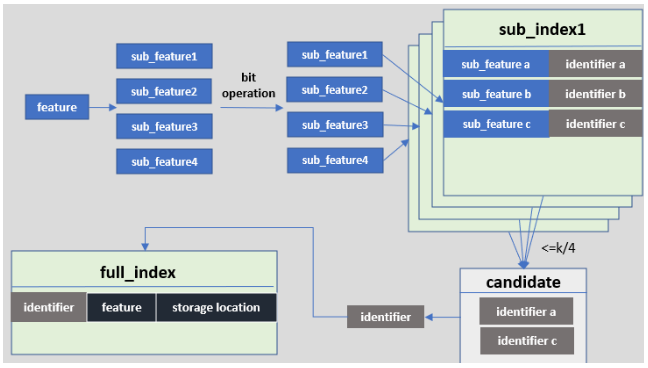

Figure 1: Search process in Multi Index.

1.2 SGX

Intel SGX [1] is a new extension of the Intel architecture, adding a new set of instructions and memory access mechanisms to the existing architecture. These extensions allow applications to implement a container called **Enclave**, which divides a protected area in the application's address space, providing confidentiality and integrity protection for the code and data inside the container from malicious software with special permissions. Enclave is a protected content container used to store application sensitive data and code. When the parts of the application that need to be protected are loaded into Enclave, SGX protects them from external software access. All enclaves reside in the **EPC** (enclave page cache), which is a protected physical memory area within the system used to store enclaves and SGX data structures.At the same time, SGX provides interface functions *ECall* and *OCall* to achieve secure interaction between enclave and unprotected memory. SGX has been supported and widely deployed by Intel CPUs of various models, and is a natural selection for data query security protection.

Limitations of SGX. We will analyze the current limitations of SGX from two aspects: (i) the limited size of the enclave, and (ii) cost of crossing enclave boundary. We know that enclave created in EPC can provide confidentiality and integrity for applications run in it, but the EPC size is limited to 128MB [3, 4](SGXv1, SGXv2 [2] supports a large EPC but loses the integrity guarantee). This means that when an enclave reaches the EPC size, the memory page needs to be swapped out or in from the EPC, SGX needs to encrypt and decrypt the page, which will result in significant additional latency [6–8]. On the other hand, because enclave can't make system calls, any system service needs to exit enclave, which is very expensive.Studies have shown that the cost is about

8000 cycles [14, 19]. Entering and exiting an enclave needs hardware operations and EPC fault handling due to the EPC limit also requires exiting the enclave. [9] So our design must focus on how to solve the problems of limited EPC space and cost of crossing enclave boundary.

1.3 Problem

1.3.1 Application Scenario

C/S Architecture. We propose a secure data query scheme based on SGX, which adopts a C/S architecture and protects all features by maintaining indexes in the server's enclave. Firstly, the server establishes an index table based on the feature value dataset in the enclave. Secondly, the client communicates with the server through a secure channel. Then, the client initiates a query request to the server based on feature values and thresholds. Lastly, the server completes the search in enclave and returns the query result to the client. It agree a session key between the client and the enclave via Diffie-Hellman key exchange, which ensures the secure communication between client and server.

Application Scenario. The application scenario of this system can be scenarios that require similarity queries on multimedia sensitive data, such as image recognition, using the security protection provided by SGX to protect the privacy and integrity of sensitive data. In these application scenarios, we can convert multimedia data into binary features and use Multi Index to store features. By utilizing the security protection provided by SGX, we can store the data and code of feature values in the enclave to prevent attackers from accessing and changing with them. We also perform feature similarity search in the enclave to prevent attackers from obtaining information by observing customer query behavior. At the same time, SGX can also protect enclaves from physical attacks, improving system security and reliability.

1.3.2 Threat Model

This security search system runs in the enclave of SGX, and its trusted computing base (TCB) is the processor chip and the code running in the enclave, so it can resist malicious privilege attacks and some physical attacks, and it does not rely on the security of the operating system. In addition, it can resist conventional physical attacks, such as bus detection attacks [15] and cold start attacks [**?**]. However, currently SGX is unable to withstand side channel attacks [5, 16]. The client and the server exchange keys through Diffie-Hellman, which makes it impossible for both parties to infer the session key. The client submits a query request through a secure channel protected by a key. We believe that enclave is fully trusted, and this article does not consider side channel attacks.

2 Baseline

In this section, We described how to implement Multi index in SGX and the problems exposed.

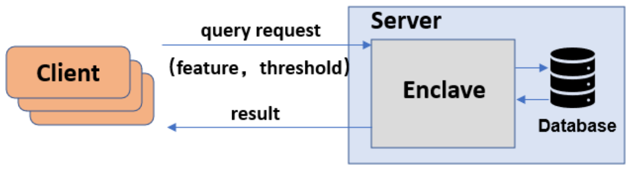

Figure 2: C/S architecture.

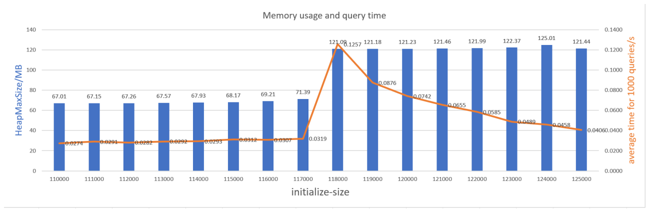

Figure 3: query performance changes as data increases

2.1 Multi-Index on SGX

Figure 2 presents the architecture of the system, in which the server maintains an enclave to store features and execute query request. The server provide an interface that allow client to submit query request and the hamming distance threshold. After completing the search in enclave,the server returns the query result to the client.

We set the feature to 128 bits, divided into 4 segments, each segment is 32 bits, Hamming distance threshold k is determined by the client's query request.Accordingly, in the enclave, we set up 4 sub feature hash tables(sub index), 1 full feature hash table(full index), and 1 candidate pool. Hash tables are all implemented with hopscotch map, and candidate pool is implemented with vector. The sub index is used to hold 4 sub feature and identifiers, and the KV structure is: key-sub feature(32bit), value-identifier(32bit). Full index is used to hold identifiers and feature, and KV is structured as: key-identifier(32bit), value-feature(128bit). Candidate pool is used to hold identifier(32bit) that the corresponding feature in full index may be similar to query q.

The whole process of a query is: (i).Client and server exchange keys through DH algorithm to establish a secure channel; (ii).The client encrypts the information and sends a query request (including the feature and threshold with query); (iii).After the server encrypts the information, it performs segmentation and other processing according to the query request; (iiii). carrying out similarity search; (iiiii).Encrypt the result and return it through the secure channel.

We tested the system with real data as set up above. Shown as Figure 3, test results showed a significant change in total query time (1000 queries) when the memory usage of the enclave was around 90 MB.We think this is due to the EPC size limit of 128MB. At this point, the system's paging costs become non-negligible.

2.2 Limitation

We will illustrate the limitations of the system from two perspectives.

- **Many invalid queries.** In each sub index, all hash buckets with Hamming distances no more than [k/m] from the query q need to be traversed. However, most of the hash buckets are empty, and these queries are unnecessary.
- **Limited EPC size.** We observe that using hash maps to achieve multiple indexes can improve computational efficiency, but at the expense of space, which is not friendly to space-constrained EPC. As shown in Figure 3, when the memory usage reaches about 90MB, the query time varies rapidly due to the page-changing overhead of the EPC.

The following three experiments help illustrate the limitations mentioned above.

References

[1] Intel(R) software guard extensions. `https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html`.

[2] Supporting intel sgx on multi-socket platforms. `https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions/supporting-sgx-on-multi-socket-platforms.html`.

[3] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. Stillwell, et al. Scone: Secure linux containers with intel SGX. In *Proc. of USENIX OSDI*, 2016.

[4] M. Bailleu, J. Thalheim, P. Bhatotia, C. Fetzer, M. Honda, and K. Vaswani. SPEICHER: Securing lsm-based key-value stores using shielded execution. In *Proc. of USENIX FAST*, 2019.

[5] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang. Detecting privileged side-channel attacks in shielded execution with déjá vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 7–18, 2017.

[6] T. Dinh Ngoc, B. Bui, S. Bitchebe, A. Tchana, V. Schiavoni, P. Felber, and D. Hagimont. Everything you should know about Intel SGX performance on virtualized systems. In *Proc. of ACM SIGMETRICS*, 2019.

[7] M. El-Hindi, T. Ziegler, M. Heinrich, A. Lutsch, Z. Zhao, and C. Binnig. Benchmarking the second generation of Intel SGX hardware. In *Proc. of Data Management on New Hardware*, 2022.

[8] D. Harnik, E. Tsfadia, D. Chen, and R. Kat. Securing the storage data path with SGX enclaves. `https://arxiv.org/abs/1806.10883`, 2018.

[9] T. Kim, J. Park, J. Woo, S. Jeon, and J. Huh. Shieldstore: Shielded in-memory key-value storage with sgx. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–15, 2019.

[10] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2064–2072, 2016.

[11] Q. Liu, Y. Shen, and L. Chen. Hap: An efficient hamming space index based on augmented pigeonhole principle. In *Proceedings of the 2022 International Conference on Management of Data*, pages 917–930, 2022.

[12] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. *mij*, 1:2, 2011.

[13] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3108–3115. IEEE, 2012.

[14] M. Orenbach, P. Lifshits, M. Minkin, and M. Silberstein. Eleos: Exitless os services for sgx enclaves. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 238–253, 2017.

[15] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard. Drama: Exploiting dram addressing for cross-cpu attacks. In *USENIX Security Symposium*, pages 565–581, 2016.

[16] M.-W. Shih, S. Lee, T. Kim, and M. Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *NDSS*, 2017.

[17] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):66–78, 2012.

[18] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. *Advances in neural information processing systems*, 21, 2008.

[19] O. Weisse, V. Bertacco, and T. Austin. Regaining lost cycles with hotcalls: A fast interface for sgx secure enclaves. *ACM SIGARCH Computer Architecture News*, 45(2):81–93, 2017.

[20] S. Zhao, D. Wu, W. Zhang, Y. Zhou, B. Li, and W. Wang. Asymmetric deep hashing for efficient hash code compression. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 763–771, 2020.