POSTER: Accelerating Encrypted Data Stores Using Programmable Switches

Carson Kuzniar*, Miguel Neves†, Israat Haque*
Dalhousie University*, UFRGS†

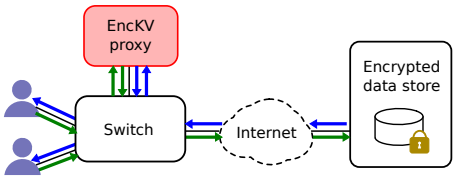
***Abstract*—This poster presents P4-EncKV, an in-network proxy for accelerating encrypted data stores using recent programmable switches. P4-EncKV can perform operations over encrypted data while reducing query latency and required bandwidth. As proof-of-concept, we implement a prototype of P4-EncKV using BMv2 software switch, and show it can speed-up encrypted queries by 20-25% using basic caching operations. Our optimized cache design also reduces memory consumption by 18% compared to the state-of-the-art in-network caching approach, thanks to a novel hash-based indexing scheme.**

I. Introduction

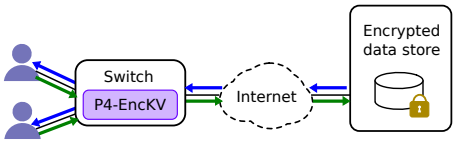
Cloud computing has become ubiquitous due to its elasticity, availability, and economies of scale (large cloud providers can amortize operational costs among multiple tenants). However, many organizations are still reluctant to use third-party clouds because often their applications require computations over sensitive data. A rich body of work has been devoted to enabling cloud applications, and in particular databases (or data stores), to work transparently on top of ciphertexts, e.g., by using specialized encryption schemes [1] or trusted execution environments [2]. Nevertheless, current approaches either exhibit significant performance penalties or are limited in terms of their operations. In common, most of them require routing queries over a proxy responsible for performing ciphertext operations (caching, summing or counting) without modifying traditional database servers (e.g., MongoDB, PostgreSQL) [1], as shown in Figure 1a.

In this poster, we propose to accelerate encrypted database queries by leveraging new programmable switches. These switches can provide flexible traffic processing up to multiple Tbps and naturally sit between database servers and clients. We explore their high-speed and flexibility to design an *in-network proxy for encrypted key-value stores*, called P4-EncKV (Figure 1b). Our system can speed-up queries (in particular read queries) while saving bandwidth capacity and preserving existing security properties, e.g., bounded information leakage. In this preliminary effort, we describe how P4-EncKV performs caching operations over encrypted data, and leave a more detailed discussion on how to perform further processing (e.g., homomorphic addition, oblivious access) as future work.

Unlike previous in-network caching approaches [3], which target plaintext key-value stores (though could be adapted to encrypted ones with some engineering effort), we support storing values at any granularity. Together with our novel hash-based indexing scheme, the P4-EncKV design significantly reduces the extent of fragmentation on the constrained switch memory resources. We deploy a prototype of P4-EncKV on BMv2 software switch [4] and demonstrate that it can accelerate encrypted database queries by 20-25% while reducing memory usage by 18% compared to the state-of-the-art in-network caching approach.



(a) Existing encrypted key-value stores.



(b) P4-EncKV

Figure 1: P4-EncKV is a re-design of encrypted data stores.

Related work. There have been many efforts exploiting programmable switches to accelerate applications. Examples include spoofed traffic filters [5], coordination services [6], and plain text database queries [7], [8]. In contrast, we take the first step toward accelerating encrypted applications (i.e., applications that do not require traffic to be decrypted at rest, processing, or transport).

II. Towards a network accelerated encrypted key-value store

We deploy an in-network proxy (P4-EncKV) to accelerate encrypted key-value stores. In this poster, we focus on describing how P4-EncKV performs encrypted caching operations. Figure 2 shows our design. It uses a set of register arrays to store variable-length values, and match-action tables to access the appropriate index in each array to reconstruct the value content. For example, the value associated with *encrypted key* $E(C)$ is the concatenation of blocks at indexes 2 and 0 (the blue ones) in arrays 1 and 2, respectively. Unlike previous work [3], we define register sizes in a power of 2 basis, which

978-1-7281-6992-7/20/$31.00

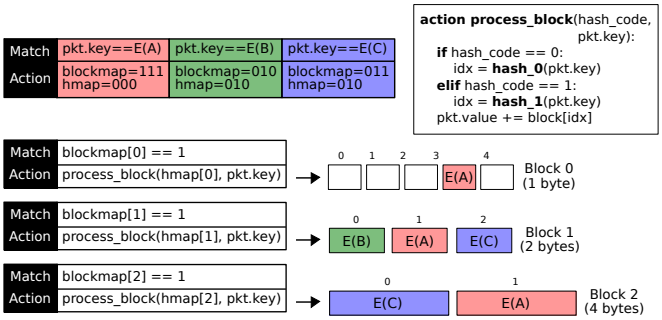


Figure 2: Encrypted key-value caching in the data plane.

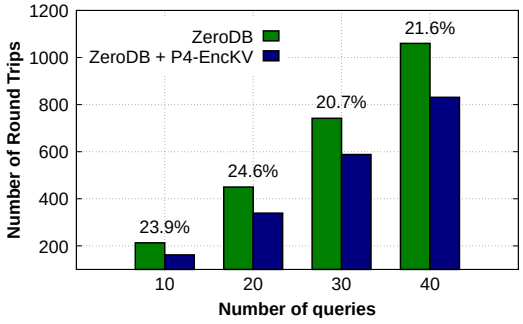


Figure 3: Total number of round trips when varying the number of queries. Values on top show the percentage gains of P4-EncKV relative to traditional ZeroDB.

	P4-EncKV	NetCache
Memory (Kbytes)	1.64	2.00

Table I: Switch resource usage to cache top layers of ZeroDB index tree. Maximum size for cached values is set to 400 bytes.

allows us to serve values *at any granularity*. Moreover, a hash-based indexing scheme enables us to have variable-length arrays (i.e., arrays with different numbers of elements) and thus save precious memory in the network device by reducing fragmentation.

We use an additional lookup table to match the encrypted key and generate two bitmaps (i.e., metadata). A *block map* indicating the arrays containing blocks that are part of the desired value and a hash map (*hmap*) that encodes a hash function to find an index inside a given array. In this case, we are *neither limited to storing all blocks of a given value on the same index, nor need a list of indexes to find the right position for each array*. Different hash functions can be used to avoid collisions, and the length of the hash map is proportional to the number of hash functions used.

III. Preliminary Results

We prototype P4-EncKV using the BMv2 software switch [4]. The prototype extends NetCache implementation [3] by adding a hash map for each encrypted key and adopting variable-sized register blocks. We use Mininet (version 2.3.0) and the ZeroDB encrypted database (version 0.98.0) [2] to test our solution. ZeroDB enables clients to encrypt data and their indexes before outsourcing both to an untrusted server. This process ensures that the server never knows the content of data objects or which objects belong to a given index. The server stores indexes in an encrypted B-Tree that clients need to traverse over a series of round trips to retrieve the requested encrypted records (each round trip corresponds to the next level in the B-Tree). We use P4-EncKV to cache ZeroDB indexes, so on a cache hit the information is retrieved from the switch instead of a complete round trip to the server. Because all traffic passes through the switch, on a cache miss there is no impact on performance.

Our experiments consider a database filled with 5K records (random integer numbers), and we use a Python controller to populate our key-value store with the most frequent indexes (top levels of the B-Tree). Our results show that P4-EncKV can achieve 20-25% improvements in the total number of round trips for equality queries (Figure 3). In addition, P4-EncKV reduces memory usage by 18% compared to NetCache (Table I), as it minimizes the fragmentation issue originated from the NetCache register indexing scheme.

IV. Conclusion and Future Work

In this poster, we begin to explore the potential for programmable switches to speed up queries over encrypted data stores. We propose P4-EncKV, an in-network proxy that can reduce query latency and bandwidth while preserving existing security properties. Our proof-of-concept prototype shows it is possible to reduce query latency by 20-25% with simple caching operations. As ongoing work, we are extending P4-EncKV to support more complex operations such as oblivious access and homomorphic encryption arithmetic. Future efforts also include evaluating our system on hardware switches.

References

[1] R. Poddar, T. Boelter, and R. A. Popa, "Arx: An encrypted database using semantically secure encryption," *Proc. VLDB Endow.*, vol. 12, no. 11, p. 1664–1678, Jul. 2019.

[2] M. Egorov and M. Wilkison, "Zerodb white paper," 2016.

[3] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17, 2017, p. 121–136.

[4] P. L. Consortium, *P4 Behavioral Model*, 2015. [Online]. Available: https://github.com/p4lang/behavioral-model

[5] G. Li, M. Zhang, C. Liu, X. Kong, A. Chen, G. Gu, and H. Duan, "Nethcf: Enabling line-rate and adaptive spoofed ip traffic filtering," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, 2019, pp. 1–12.

[6] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica, "Netchain: Scale-free sub-rtt coordination," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 35–49.

[7] M. Tirmazi, R. Ben Basat, J. Gao, and M. Yu, "Cheetah: Accelerating database queries with switch pruning," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20, 2020, p. 2407–2422.

[8] A. Lerner, R. Hussein, and P. Cudré-Mauroux, "The case for network accelerated query processing," in *CIDR*, 2019.