Accelerating Encrypted Deduplication via SGX

Yanjing Ren*, Jingwei Li*, Zuoru Yang#, Patrick P. C. Lee#, and Xiaosong Zhang*

*University of Electronic Science and Technology of China
#The Chinese University of Hong Kong

USENIX ATC 2021
Outsourcing Storage

➢ Outsourcing data management to cloud is common in practice
  • 22% business data are stored in the cloud[*]

➢ Outsourcing storage should fulfill security and storage efficiency
  • Security: protect outsourced data against unauthorized access
  • Storage efficiency: reduce storage footprints

Encrypted Deduplication

- Encrypt plaintext chunks followed by performing deduplication on ciphertext chunks
  - Traditional encryption is incompatible with cross-user deduplication

- Message-locked encryption (MLE) [Bellare, Eurocrypt’13]: use content-derived keys for encryption, so as to enable cross-user deduplication

![Diagram]

**Example:**

\[ K = \text{hash of plaintext} \]
MLE-based Implementation

➢ Use **server-aided architecture** to prevent offline brute-force attacks

➢ Protect key generation via **oblivious pseudorandom function (OPRF)** to prevent key server from learning plaintext chunks

➢ Perform **target-based** [Bellare, Security’13] or **source-based** [Halevi, CCS’11] deduplication
  - Target-based: upload all chunks and remove duplicates in the cloud
  - Source-based: upload fingerprints for duplicate check, followed by only non-duplicate chunks
MLE-based Implementation

➢ OPRF is known to incur **high computational overhead** [Qin, TOS’17]

➢ Target-based deduplication has **high bandwidth overhead**

➢ Source-based deduplication incurs information leakage
  • A malicious client can fake fingerprints to learn deduplication patterns of corresponding chunks
  • Need to be protected by **proof-of-ownership (PoW)** [Halevi, CCS’11], which is **computationally expensive**
MLE-based Implementation

➢ OPRF is known to incur high computational overhead [Qin, TOS’17]
➢ Target-based deduplication has high bandwidth overhead
➢ Source-based deduplication incurs information leakage
   • A malicious client can fake fingerprints to learn deduplication patterns of corresponding chunks
   • Need to be protected by proof-of-ownership (PoW) [Halevi, CCS’11], which is computationally expensive

How to accelerate encrypted deduplication while preserving security?
Contributions

➢ **SGXDedup**: use *Intel SGX* to speed up encrypted deduplication
  • Replace expensive cryptographic protection by *hardware-based protection*
  • Three key designs to preserve security and boost performance

➢ Extensive experiments:
  • **131.9×** key generation and **8.2×** PoW speedups over existing approaches
  • **8.1×** throughput over existing software-based encrypted deduplication [Bellare, Security’13]
SGX Basics

➢ **Isolation**: allow to allocate an isolated memory region (enclave) against host system
  • Enclave is of limited size (e.g., 128MB)

➢ **Attestation**: can attest in-enclave contents via remote attestation
  • Remote attestation incurs huge latency (e.g., ~9s in our region)

➢ **Sealing**: enclave can securely move in-enclave contents into unprotected memory via encryption
  • Only the same enclave can access its sealed contents
Design Goals

➢ Preserve goals of software-based encrypted deduplication
  • Confidentiality: Protect chunks and keys against unauthorized access
  • Storage efficiency: Remove all duplicate chunks

➢ Boost performance via hardware-based approach
  • Bandwidth efficiency: Only need to transfer non-duplicate chunks
  • Computational efficiency: Mitigate computational overhead of cryptographic primitives
SGXDedup

Key enclave:
- Connected with each client via secure channel
- Perform key generation: $K = H(fp \ || \ GlobalSecret)$

PoW enclave:
- Generate signature for each fingerprint, such that cloud can verify authenticity of fingerprints → lightweight protection on source-based deduplication
Questions

➢ Q1: How should enclaves be securely and efficiently bootstrapped?
   • The global secret needs to be securely bootstrapped into key enclave
   • Enclave startup incurs high latencies due to remote attestation

➢ Q2: How should the secure channel be established?
   • Necessary to enable revocation on clients’ querying key generation

➢ Q3: How should key enclave reduce its computational overhead of managing secure channels?
   • The computational overhead is high as the number of clients increases
Enclave Management

➢ Compute global secret from an in-enclave sub-secret (from cloud) and an input sub-secret (from key server)
   • Prevent either cloud or key server from learning the whole global secret

➢ Attest key enclave and PoW enclave offline
   • After attestation, both cloud and each PoW enclave share a PoW key to verify authenticity of fingerprints

➢ Use sealing to avoid re-attesting PoW enclave after its first bootstrap
   • PoW enclave may be bootstrapped and terminated with client
   • Seal (unseal) PoW key when PoW enclave terminates (bootstraps again)
Renewable Blinded Key Management

- Build secure channel based on a **blinded key** shared by clients and key enclave
- Update blinded key if some clients are revoked
  - Key update is based on key regression \([\text{Fu, NDSS’06}]\), so as to support lazy update
- Synchronize blinded keys between key enclave and authorized clients
  - Key enclave derives new blinded keys based on an in-enclave **blinded secret**
  - Authorized clients download up-to-date blinded keys from cloud
SGX-based Speculative Encryption

➢ Build on **speculative encryption** [Eduardo, FAST’19] to reduce online computational overhead of key enclave
  • Speculative encryption: \(fp \ XOR E(\text{blindedKey, nonce}||\text{counter})\) mask
  • Allow to compute masks offline

➢ Manage each nonce and corresponding masks in key enclave
  • Each client is associated with a nonce
  • Manage an in-enclave **nonce index** to ensure unique nonce for each client
  • Take up to 3MB enclave space for nonce index to serve 112K clients

➢ Pre-compute masks of each nonce automatically
  • Store pre-computed masks in a 90MB **mask buffer** that can be used to process the fingerprints of 11.25GB data
Experimental Setup

➢ Implement SGXDedup in Linux
  • ~14.2K line of C++ code

➢ Real-world datasets:
  • FSL: users’ home directory backups (56.2TB, 431.9GB after deduplication)
  • MS: windows file system snapshots (14.4TB, 2.4TB after deduplication)

➢ Testbed:
  • Multiple machines connected with 10GbE
  • Each machine has Intel Core i5-7400 3.0GHz CPU and 8GB RAM
Overall System

➢ **8.1x** and **9.6x** speedups over DupLESS in first and second uploads
  • The performance of DupLESS is bounded by OPRF-based key generation
  • The second upload is faster than the first upload due to source-based deduplication

➢ **17.5%** upload and **44.2%** download performance drops over PlainDedup
  • Overhead comes from key generation, encryption, PoW and decryption

➢ More results in our paper:
  • **637.0 MB/s** aggregate upload speed for 10 clients
  • **9.7x** speedup over DupLESS in real-cloud deployment
Trace-driven Performance

- SGXDedup incurs **21.4%** upload performance drop from PlainDedup
  - To replay trace, chunking is disabled
  - The bottleneck of SGXDedup is PoW while that of PlainDedup is fingerprinting

- The download speed is bounded due to chunk fragmentation
Conclusion

➢ **SGXDedup**: mitigate performance overhead of encrypted deduplication via SGX
  - Offload expensive cryptographic operations by directly running sensitive operations in enclaves
  - Three designs:
    - Secure and efficient enclave management
    - Renewable blinded key management
    - SGX-based speculative encryption for lightweight computations

➢ Source code: [http://adslab.cse.cuhk.edu.hk/software/sgxdedup](http://adslab.cse.cuhk.edu.hk/software/sgxdedup)