Information Leakage in Encrypted Deduplication via Frequency Analysis

Introduction

Encrypted deduplication seamlessly combines encryption and deduplication to simultaneously achieve both data security and storage efficiency. State-of-the-art encrypted deduplication systems mostly adopt a deterministic encryption approach that encrypts each plaintext chunk with a key derived from the content of the chunk itself, so that identical plaintext chunks are always encrypted into identical ciphertext chunks for deduplication. However, such deterministic encryption inherently reveals the underlying frequency distribution of the original plaintext chunks. This allows an adversary to launch frequency analysis against the resulting ciphertext chunks, and ultimately infer the content of the original plaintext chunks.

We study how frequency analysis practically affects information leakage in encrypted deduplication storage, from both attack and defense perspectives. We first propose a new inference attack that exploits chunk locality to increase the coverage of inferred chunks. We conduct trace-driven evaluation on a real-world dataset, and show that the new inference attack can infer a significant fraction of plaintext chunks under backup workloads. To protect against frequency analysis, we borrow the idea of existing performance-driven deduplication approaches and consider an encryption scheme called MinHash encryption, which disturbs the frequency rank of ciphertext chunks by encrypting some identical plaintext chunks into multiple distinct ciphertext chunks. Our trace-driven evaluation shows that MinHash encryption effectively mitigates the inference attack, while maintaining high storage efficiency.

The toolkit is used to simulate the attack and defense approaches based on the fsl trace that consists of fslhome snapshots.

Publication

- Jingwei Li, Chuan Qin, Patrick P. C. Lee, Xiaosong Zhang. Information Leakage in Encrypted Deduplication via Frequency Analysis. In Proc. of IEEE/IFIP DSN, 2017.

Preparation

The attack is running under Linux (e.g., Ubuntu 14.04) with a C++ compiler (e.g., g++). To run the attack program, you need to install/compile the following dependencies.

- Libssl API: run the command `sudo apt-get install libssl-dev`.
- Snappy compression library: run the command `sudo apt-get install libsnappy-dev`.
- Google Leveldb: a version of 1.20 is provided in `util/`
- fs-hasher: a version of 0.9.4 is provided in `util/`

Attacks

Basic Attack

The basic attack builds on classical frequency analysis. Follow the steps to simulate the basic attack.

Step 1, configure pre-requsite components: copy `util/fs-hasher/` and `util/leveldb/` into `attack/basic/` and compile them respectively.

Step 2, configure basic attack: modify variables in `attack/basic/basic_script.sh` to adapt expected settings:

- `fsl` specifies the path of the fsl trace.
- `users` specifies which users are collectively considered in backups.
- `date_of_aux` specifies the backup of which date is considered as auxiliary information.
- `date_of_latest` specifies the backup of which date is the target for inference.

Step 3, run basic attack: type the following commands to compile and run the basic attack.

```
$ cd attack/basic/
$ make
$ ./basic_script.sh
```

Locality-based Attack

The locality-based attack exploits chunk locality to improve attack severity. To simulate the locality-based attack, follow the steps below.

Step 1, configure pre-requsite components: copy `util/fs-hasher/` and `util/leveldb/` into `attack/locality/` and compile them respectively.

Step 2, configure locality-based attack: In addition to the common variables (e.g., `fsl`, `users`, `date_of_aux` and `date_of_latest`), the locality-based attack builds on four parameters that are defined in `attack/locality/locality_script.sh`:

- `u` specifies the number of most frequent chunk pairs to be returned by frequency analysis in initializing the inferred set.
- `v` specifies the number of most frequent chunk pairs to be returned by frequency analysis in each iteration.
- `w` specifies the maximum number of ciphertext-plaintext chunk pairs that can be held by the inferred set.
- `leakage_rate` specifies the ratio of the number of ciphertext-plaintext chunk pairs known by the adversary to the total number of ciphertext chunks in the latest backup.

Step 3, run locality-based attack: type the following commands to compile and run the locality-based attack.

```
$ cd attack/locality/
$ make
$ ./locality_script.sh
```

Defense

MinHash Encryption

To defend the locality-based attack, MinHash encryption derives an encryption key based on the minimum fingerprint over a set (called segment) of adjacent chunks (that are assumed to be with an average size of 8KB), such that some identical plaintext chunks can be encrypted into multiple distinct ciphertext chunks. To simulate the MinHash encryption, follow the steps below.

Step 1, configure pre-requsite components: copy `util/fs-hasher/` and `util/leveldb/` into `defense/minhash/` and compile them respectively.

Step 2, configure MinHash encryption: the MinHash encryption builds on two parameters that are defined in `defense/minhash/k_minhash.cc`:

- **Segment size**: the MinHash implementation uses variable-size segmentation and identifies segment boundary based on chunk fingerprints. By default, we set the average segment size, maximum segment size and minimum segment size at 1MB, 2MB and 512KB, respectively. It is feasible to change segment sizes by modifying macro variables `SEG_SIZE`, `SEG_MIN` and `SEG_MAX`; note that when changing `SEG_SIZE`, it is needed to adjust the code in line 112 of `defense/minhash/k_minhash.cc`, for example if average segment size is 512KB and 2MB, the line of code should be changed as follows.

```
if (sg_size + size > SEG_MAX || (sg_size >= SEG_MIN && (hash[5] << 3) >> 3 == 0x1f))    // correspond to average segment size of 512KB
if (sg_size + size > SEG_MAX || (sg_size >= SEG_MIN && (hash[5] << 1) >> 1 == 0x7f))    // correspond to average segment size of 2MB
```

- **K**: our implementation supports k-MinHash that derives an encryption key from a random k-minimum fingerprint of a segment. By default, we use MinHash and set `K_MINHASH` by 1.

Step 3, configure locality-based attack: it is identical to Step 2 of the guideline of locality-based attack, except the attack variables (e.g., `u`, `v` and `w`) locate in `defense/minhash/defense_script.sh`.

Step 4, run MinHash encryption to defend locality-based attack: type the following commands to run.

```
$ cd defense/minhash/
$ make
$ ./defense_script.sh
```

Output Formats

The output format is shown as follows

```
==========================Attack/Defense==========================
Auxiliary information: YYYY-MM-DD;      Target backup: YYYY-MM-DD
[Parameters: (u, v, w) = ...]
Total number of unique ciphertext chunks: X
[Leakage rate: ...]
Correct inferences: Y
Inference rate: ...

Successfully inferred following chunks:
------
```

`X` is the number of unique ciphertext chunks in the encryption of the target backup, while `Y` is the number of (unique) chunks that can be successfully inferred by the attacks. The inference rate is computed by `Y/X`, that is slightly affected by the sorting algorithm in frequency analysis. The reason is different sorting algorithms may break tied chunks (that have the same frequency counts) in different ways and lead to (slightly) different results. The `parameters` and `leakage rate` are only available in the simulation of locality-based attack and its defense. We output the fingerprints of inferred plaintext chunks in both attacks and defense simulation.