Enabling Secure and Space-Efficient Metadata Management in Encrypted Deduplication (Supplementary File)

Jingwei Li, Suyu Huang, Yanjing Ren, Zuoru Yang, Patrick P. C. Lee, Xiaosong Zhang, and Yao Hao

1 Security Analysis of Metadata Chunks

In Metadedup, each data chunk remains protected by historical MLE, so the confidentiality guarantees for unpredictable data chunks are retained in the context of encrypted deduplication. However, in the security level under historical MLE, an adversary can derive metadata (e.g., keys) from data chunks and arbitrarily construct metadata chunks. Since Metadedup also protects metadata chunks using historical MLE, the adversary can launch the offline brute-force attack (see Section 2.1 of the main file) to infer the original contents in target metadata chunks.

We argue that the offline brute-force attack against encrypted metadata chunks is much more computationally expensive than against encrypted data chunks. Recall that each metadata chunk consists of the metadata of multiple encrypted data chunks. Thus, an adversary needs to include different combinations of encrypted data chunks to construct a potential encrypted metadata chunk for the offline brute-force attack, yet the number of combinations is exhaustively high. In contrast, to launch the offline brute-force attack against encrypted data chunks, the adversary only needs to sample each possible data chunk to test (see Section 2.1 of the main file).

We conduct a simple analysis to justify that the offline brute-force attack against a metadata chunk incurs a huge time cost and hence is computationally infeasible in practice. Suppose that each data chunk is known to be drawn from a finite set that includes a total of n distinct data chunks. Let c be the average number of data chunks in a segment.

To compute the metadata of an encrypted data chunk, the adversary applies a hash function once to derive the MLE key, encrypts the data chunk, and applies the hash function again to derive the fingerprint. We estimate the running time of computing the metadata of each data chunk as:

$$T_{\mathsf{meta}} = 2 \times T_{\mathsf{hash}} + T_{\mathsf{enc}},$$

where T_{hash} and T_{enc} denote the running times of the hash and encryption functions, respectively.

To construct a metadata chunk, the adversary assembles the metadata of c encrypted data chunks in order. In fact, each data chunk in the finite set may contribute metadata, and we assume that the metadata contribution of a data chunk does not affect that of any other data chunk (i.e., the events are mutually independent). Here, we consider the total number of combinations of c *distinct* encrypted data chunks as:

$$N_{\mathsf{assemble}} = \prod_{i=0}^{c-1}(n-i) = \frac{n!}{(n-c)!},$$

where $n!$ and $(n-c)!$ are the factorials of n and $n-c$, respectively. Each combination corresponds to a metadata chunk, and thus the adversary needs to test N_{assemble} possible metadata chunks to see if any of them is encrypted to the target encrypted metadata chunk based on historical MLE. Note that the adversary may test more metadata chunks than N_{assemble} in practice, in order to address the case that metadata chunks include the metadata of duplicate encrypted data chunks. Thus, N_{assemble} can be viewed as the *lower bound* of the number of metadata chunks that the adversary needs to construct, and the total construction time is:

$$T_{\mathsf{construct}} = N_{\mathsf{assemble}} \times T_{\mathsf{meta}}.$$

For each constructed metadata chunk, the adversary checks whether it is the original input of the target encrypted metadata chunk, and each check requires one hash (to derive the MLE key) and one encryption. This implies that the running time of the equality check for all metadata chunks:

$$T_{\mathsf{check}} = N_{\mathsf{assemble}} \times (T_{\mathsf{hash}} + T_{\mathsf{enc}}).$$

We can now estimate the average running time of the offline brute-force attack against metadata chunks as:

$$T_{\mathsf{attack}} = T_{\mathsf{construct}} + T_{\mathsf{check}}.$$

We consider an example to understand how large T_{attack} is. Suppose that the average segment size is 1 MB, and the average chunk size is 8 KB. Then, $c = \frac{1\,\mathrm{MB}}{8\,\mathrm{KB}} = 128$. According to [1], we assume that it takes T_{enc} = 48μs and T_{hash} = 37μs to perform the encryption and hash operations on a chunk of 8 KB, respectively (the equivalent encryption and hash speeds are 163 MB/s and 212 MB/s, respectively). We estimate T_{attack} as follows.

$$\begin{aligned} T_{\mathsf{attack}} &= (3 \times T_{\mathsf{hash}} + 2 \times T_{\mathsf{enc}}) \times \frac{n!}{(n-c)!} \\ &\geq (3 \times T_{\mathsf{hash}} + 2 \times T_{\mathsf{enc}}) \times c! \\ &\approx 7.94 \times 10^{211} s. \end{aligned}$$

TABLE 1: Overall storage efficiency of MD under different average segment sizes. *Raw* denotes the original metadata size without MD.

	Components/Metrics	Raw	512KB	1MB	2MB	4MB
FSL	File recipes (GB)	178.191	1.932	0.965	0.481	0.240
	Key recipes (GB)	190.070	2.061	1.030	0.513	0.256
	Fingerprint index (GB)	1.385	1.412	1.400	1.393	1.390
	Metadata chunks (GB)	–	6.806	7.372	8.041	8.818
	Total metadata size (GB)	**369.646**	**12.211**	**10.767**	**10.428**	**10.704**
	Metadata storage saving	**–**	**97.07%**	**97.46%**	**97.55%**	**97.47%**
	Index overhead	**–**	**1.94%**	**1.07%**	**0.60%**	**0.33%**
VM	File recipes (GB)	297.070	1.192	0.596	0.299	0.147
	Key recipes (GB)	316.875	1.271	0.636	0.319	0.157
	Fingerprint index (GB)	1.230	1.269	1.259	1.254	1.251
	Metadata chunks (GB)	–	8.106	9.793	12.276	16.824
	Total metadata size (GB)	**615.175**	**11.838**	**12.284**	**14.148**	**18.379**
	Metadata storage saving	**–**	**98.28%**	**98.20%**	**97.90%**	**97.21%**
	Index overhead	**–**	**1.91%**	**1.12%**	**0.69%**	**0.44%**
MS	File recipes (GB)	59.344	0.595	0.300	0.152	0.077
	Key recipes (GB)	63.300	0.634	0.320	0.162	0.083
	Fingerprint index (GB)	9.696	9.849	9.780	9.741	9.721
	Metadata chunks (GB)	–	32.376	34.933	37.462	39.934
	Total metadata size (GB)	**132.339**	**43.454**	**45.332**	**47.518**	**49.814**
	Metadata storage saving	**–**	**72.60%**	**71.01%**	**69.20%**	**67.31%**
	Index overhead	**–**	**1.58%**	**0.87%**	**0.47%**	**0.26%**

If the attack is implemented serially, the total running time is at least $7.94 \times 10^{211} s$, which is more than 10^{204} years. Even the attack can be implemented in parallel, it is computationally infeasible to work in reasonable time.

2 Storage Efficiency of Metadata Deduplication

We study the metadata deduplication (MD) approach (see Section 4.2 of the main file). Our goal is to show that MD significantly suppresses the storage space of metadata, while incurring limited indexing penalties. Note that MD does not change the underlying data deduplication mechanism, and we do not consider data storage saving here.

Experiment C.1 (Overall storage efficiency). We now evaluate the overall storage efficiency of MD. In addition to FSL and VM, we include the *MS* dataset for cross-dataset validation. Specifically, MS is collected by Microsoft [8], and contains 857 Windows file system snapshots. We focus on an average chunk size of 8 KB, and choose 143 snapshots, whose sizes are between 100 GB and 110 GB. This dataset takes 14.73 TB of data before deduplication.

Table 1 presents the simulation results of metadata storage saving and index overhead after storing all backups/snapshots, where *raw* denotes the original metadata size without deduplication or compression.

For the FSL dataset, the metadata storage saving first increases with the average segment size, because a larger segment size (and hence larger metadata chunks) reduces the metadata of metadata chunks. When the average segment size is 2 MB, the FSL dataset achieves the highest metadata storage saving of 97.55%. Then, the metadata storage saving decreases (e.g., in the VM dataset) due to a small deduplication factor for large metadata chunks. Nevertheless, the storage savings under all average segment sizes are higher than 97%. In addition, the index overheads decrease with the average segment size and are lower than 2%.

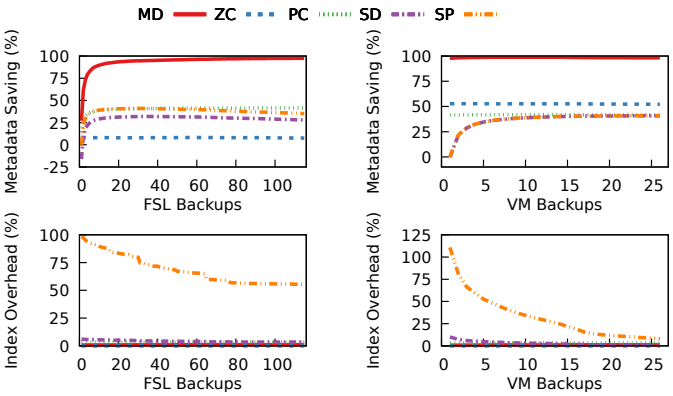

Fig. 1: Comparison of MD with compression approaches on metadata storage saving and index overhead.

The metadata storage saving of the MS dataset is lower than that of the FSL and VM datasets. The reason is that the MS dataset is likely to be snapshotted from different machines, and has a smaller deduplication factor than the backup workloads (e.g., FSL and VM) that are periodically snapshotted from the same source. On the other hand, MD achieves a medium metadata storage saving (e.g., at least 67%) for the generic MS workloads, while keeping index overhead small (e.g., below 1.6%).

Experiment C.2 (Comparison of metadata storage). We fix the average segment size of MD at 1 MB, and compare its metadata storage saving and index overhead with those of file recipe compression approaches [7]. We do not consider other approaches for comparison, as they either require the deduplication information of chunks [2], [4], [6], [10] that leads to side-channel leakage in encrypted deduplication or add additional metadata [5] and data [9], [11] overheads (see Section 3 of the main file). We elaborate how we configure the baseline compression approaches [7], followed by the evaluation results.

- *Zero compression (ZC)* replaces the metadata of zero-filled chunks by one-byte special codes, so as to reduce the sizes of file recipe and key recipe.
- *Page-based compression (PC)* assumes the availability of fingerprint index, and replaces the deduplication metadata of each chunk by a codeword derived from its index offset. We configure the length of each codeword at 4 bytes [7].
- *Statistical directory (SD)* encodes the deduplication metadata of low-entropy chunks by fixed-size codewords. Like the prior work [7], we derive the entropy information from the first backup, and allocate each codeword with 3 bytes.
- *Statistical prediction (SP)* exploits the locality of neighboring chunks under the logical order. For each chunk, it stores u codewords that are mapped from the deduplication metadata of the most likely neighbors of this chunk. For compression, it replaces the deduplication metadata of these neighbors by corresponding codewords. Like the prior work [7], we derive the neighboring information from the first backup, and set u to 2.

Figure 1 shows the cumulative metadata storage savings and index overheads of all considered approaches for the FSL and VM datasets. The metadata storage savings of MD significantly outperform those of baseline approaches. For

example, they finally achieve 97.46% and 98.20% for the FSL and VM datasets, respectively. In the baseline approaches, the savings of ZC are almost unchanged, such as 7.58-8.29% for the FSL dataset and 52.14-52.74% for the VM dataset. The savings of PC gradually increase during backup periods, since some metadata in later backups have already been encoded, and they finally achieve 41.51% and 41.71% for the FSL and VM datasets, respectively. SD and SP only have savings after the initial backups, since they need to first extract the entropy and neighboring information, respectively [7]. One special note is that SD even incurs additional overheads of 16.11% and 0.42% for the initial FSL and VM backups, respectively. The reason is that it maintains a codeword index to map assigned codewords back to the corresponding deduplication metadata [7]. Such overhead can be covered in following backups, and SD finally achieves the savings of 27.99% and 40.79% for the FSL and VM datasets, respectively. The corresponding final savings of SP also reach 35.20% and 40.81%, respectively.

In addition, we observe that MD, PC, and ZC incur low index overheads, such as less than 3.33% in both datasets, during the whole backup time. SD and SP incur relatively high index overheads in initial backups, since they need to store some fingerprint-to-codeword mappings in the fingerprint index. For example, SP stores u mappings in each fingerprint index entry to map the fingerprints of some chunks that are most likely to come after the corresponding chunk to short codewords; this leads to the index overheads of 97.34% and 110.76% for the FSL and VM datasets, respectively. Such overhead can be amortized in following backups. The index overheads of SP finally decrease to 55.35% and 8.04% for the FSL and VM datasets, respectively, while those of SD drop down to 3.38% and 0.73%.

Experiment C.3 (Combined with compressions). We now examine the effectiveness of combining MD with the baseline compression approaches [7] to reduce the size of recipes. We focus on two combined approaches: (i) MD + ZC and (ii) MD + PC, which apply MD first and then use ZC and PC to suppress the metadata of metadata chunks, respectively. We do not consider other combination options as they either incur high index overhead (e.g., combined with SP) or lead to small storage savings (e.g., combined with SD). We fix the average segment size of MD as 1 MB.

Table 2 presents the simulation results after storing all backups, and we also include the results of MD only for reference. By combining MD with ZC, we can further reduce the sizes of both file recipe and key recipe, from 0.97 GB and 1.03 GB to 0.92 GB and 0.98 GB in the FSL dataset, as well as from 0.60 GB and 0.64 GB to 0.29 GB and 0.31 GB in the VM dataset, respectively. Such recipe reduction is more effective (e.g., about 50%) for the VM dataset, as VM images include large regions of zero chunks [3]. This only leads to negligible storage savings of metadata, such as 0.02% and 0.11% for the FSL and VM datasets, respectively. Similarly, the combination of MD and PC brings few additional metadata storage savings by about 0.22% for the FSL dataset and 0.09% for the VM dataset.

Our results suggest that MD can only be marginally improved by compression approaches, as compression cannot apply to the physical metadata chunks that take more than 60% of overall metadata in MD (see Table 1). Thus, MD itself sufficiently achieves high storage saving of metadata.

TABLE 2: Combination of MD with compression approaches.

Components/Metrics		FSL	VM
MD only	File recipes (GB)	0.965	0.596
	Key recipes (GB)	1.030	0.636
	Fingerprint index (GB)	1.400	1.259
	Metadata chunks (GB)	7.372	9.793
	Total metadata size (GB)	**10.767**	**12.284**
	Storage saving	**97.46%**	**98.20%**
	Index overhead	**1.07%**	**1.12%**
MD + ZC	File recipes (GB)	0.923	0.290
	Key recipes (GB)	0.984	0.309
	Fingerprint index (GB)	1.400	1.259
	Metadata chunks (GB)	7.372	9.793
	Total metadata size (GB)	**10.679**	11.651
	Metadata storage saving	**97.48%**	**98.31%**
	Index overhead	**1.07%**	**1.12%**
MD + PC	File recipes (GB)	0.129	0.079
	Key recipes (GB)	1.030	0.636
	Fingerprint index (GB)	1.401	1.259
	Metadata chunks (GB)	7.372	9.793
	Decoding mapping (GB)	0.017	0.013
	Total metadata size (GB)	**9.949**	**11.780**
	Metadata storage saving	**97.68%**	**98.29%**
	Index overhead	**1.11%**	**2.17%**

Note that PC needs to store page offsets in fingerprint index entries for encoding, and maintain a reverse mapping for decoding [7].

REFERENCES

[1] "Bearssl performance," https://bearssl.org/speed.html.

[2] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving duplicate elimination in storage systems," *ACM Transactions on Storage*, vol. 2, no. 4, pp. 424–448, 2006.

[3] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *Proc. of ACM SYSTOR*, 2009.

[4] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal content defined chunking for backup streams," in *Proc. of USENIX FAST*, 2010.

[5] J. Li, X. Chen, M. Li, J. Li, P. P. C. Lee, and W. Lou, "Secure deduplication with efficient and reliable convergent key management," *IEEE Transactions on Parallel Distributed Systems*, vol. 25, no. 6, pp. 1615–1625, 2014.

[6] G. Lu, Y. Jin, and D. H. Du, "Frequency based chunking for data de-duplication," in *Proc. of IEEE MASCOTS*, 2010.

[7] D. Meister, A. Brinkmann, and T. Süß, "File recipe compression in data deduplication systems," in *Proc. of USENIX FAST*, 2013.

[8] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," in *Proc. of USENIX FAST*, 2011.

[9] C. Qin, J. Li, and P. P. C. Lee, "The design and implementation of a rekeying-aware encrypted deduplication storage system," *ACM Transactions on Storage*, vol. 13, no. 1, pp. 9:1–9:30, 2017.

[10] B. Romański, L. Heldt, W. Kilian, K. Lichota, and C. Dubnicki, "Anchor-driven subchunk deduplication," in *Proc. of ACM SYSTOR*, 2011.

[11] Y. Zhou, D. Feng, W. Xia, M. Fu, F. Huang, Y. Zhang, and C. Li, "SecDep: A user-aware efficient fine-grained secure deduplication scheme with multi-level key management," in *Proc. of IEEE MSST*, 2015.