Efficient Encodings for Privacy-Preserving Data Storage and Transmission

Arghya Kusum Das
Department of Computer Science
University of Alaska Fairbanks
Fairbanks, AL, USA
akdas@alaska.edu

M. Oguzhan Kulekci
Department of Computer Science
Indiana University Bloomington
Bloomington, IN, USA
okulekci@iu.edu

Sharma V. Thankachan
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
svalliy@ncsu.edu

2024 IEEE International Conference on Big Data (BigData) | 979-8-3503-6248-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/BigData62323.2024.10825141

***Abstract*—Secure and privacy-preserving storage of digital data typically requires encrypting it, where the retrieval will mandate its decryption. The overhead of these encryption/decryption requirements introduce some latency, which might be limiting user experience on massive volumes for real-time applications. Reducing this computational load have been studied previously by using lightweight algorithms or selective/partial encryption schemes. In this work, we propose using a recently introduced privacy-preserving coding method [1] (COCOON'2023) to reduce this load and observe that the number of encryption/decryption operations can be reduced by more than 75%, which can be a decent relief especially for real-world applications. We particularly consider privacy requirements of some applications on multimedia data, most typically images and videos.**

***Index Terms*—privacy preserving, data encoding, image/video representation, data storage, data transmission**

I. Introduction

One of the main sources of big data today is the images and videos produced at an ever-increasing pace and volume. These data are typically stored remotely in cloud systems, enabling access from anywhere at any time and reducing the risk of loss due to equipment failures, such as a corrupted disk. It is estimated that approximately 70% to 90% of cloud storage is dedicated to such multimedia files.

Today, many of us, if not all, store our private personal images and videos remotely, outside of our direct control. In addition to personal repositories, corporate data, such as surveillance camera footage, and scientific big data such as drone images [2], Geotiff images [3], further contribute to this growing volume. Another significant source is the vast amount of medical images and videos being produced. Beyond storage, the transfer of image and video files also consumes a large portion of internet traffic, with the rise of online meeting systems further adding to this.

A common concern for individuals and companies alike is the privacy and security of their image and video data during storage, transfer, or communication. Ensuring that this data remains safe from unauthorized access is crucial. Unfortunately, incidents of unauthorized access are becoming increasingly common, as reported in the daily media.

The ultimate solution to this problem is the proper encryption of image and video data, whether at rest or in transit, ensuring their security and privacy. However, once encrypted, data must be decrypted for retrieval, and the overhead of these encryption and decryption operations can be overwhelming, especially when dealing with large volumes of data. Previous research has largely focused on lightweight algorithms [4], [5] or partial/selective encryption schemes [6]–[8].

Identify applicable funding agency here. If none, delete this.

Lightweight encryption [9] aims to reduce the computational burden of traditional algorithms by removing or modifying certain operations. A typical example is the AES scheme with fewer rounds. However, such modifications can weaken the original algorithm [10]. There have also been efforts to create entirely new cryptographic algorithms, as surveyed in [11].

Selective or partial encryption [12] offers another approach, securing data by encrypting only a portion of it, thus inherently reducing computational demands. These selective algorithms have primarily been applied to image and video security, where the data has often already been entropy encoded, such as in JPEG, MPEG, or other video/image compression formats.

Typically, the sections or parameters necessary for correctly decoding the data are selected for encryption [7], [13], [14]. Deciding which sections to encrypt is the first step, followed by identifying and encrypting those regions. However, in practice, especially with entropy-encoded image or video data, this selection process—and the intelligence required to identify or mark those encrypted sections—can present challenges [8].

In this study, we explore a recent privacy-preserving data representation method, which we refer to as ”split coding” [1], as a new alternative for selective encryption of image and video files. It is important to note that this method is not limited to video or image data; it can be applied to any type of stream such as genomic and metagenomic data [15], energy meter data [16], and so on. However, we observe that compressed image or video data, typically in formats like JPEG or H.264/265, is particularly well-suited for this approach, as the *randomization* step in the split coding scheme can be safely skipped without adverse effects. We discuss the technical details in section 2, where we review the coding technique and the simplifications we propose.

979-8-3503-6248-0/24/$31.00 ©2024 IEEE

i	1	2	3	4	5	6	7	8
S_i	100	101	001	000	010	110	111	011
$S'_i = \pi(S_i)$	011	111	010	101	000	100	110	001
L_i	01	1	01	1	000	1	1	001
R_i	1	11	0	01		00	10	

Fig. 1. The split-coding of $B = 100101001000010110111011$ with $d = 3$, $q = 1$, and the permuted alphabet $\pi = \langle 5, 2, 0, 1, 3, 7, 4, 6 \rangle$ generates the left and right partitions $\langle 01101100011001, 1110010010 \rangle$.

II. Split Coding Review

The split coding [1] of an input n bit sequence $B[1..n]$ is based on three parameters as:

- **Block-size** d: The input sequence is assumed to be a sequence of $m = \lceil n/d \rceil$ d–bits long symbols as $B = S[1..m]$, where $S[i] = B[(i-1)d..id-1]$. Observe that each symbol $S[i]$ is from the alphabet $A = \{0, 1, 2, .., 2^d - 1\}$.
- **Randomization key** π: The random shuffling of the alphabet A with Fisher-Yates method [17] by using the random-number-generator seed π creates the permutation of the alphabet A as $A'[0..2^d - 1]$.
- **Select count** q: This number determines the split point in a d–bit symbol as the qth left-most set bit position.

The split coding starts with the randomization step where each symbol $j = S[i]$ is replaced with its corresponding symbol $A'[j]$ according to the generated permutation.

After the randomization, each symbol on the sequence is split into two such that all the bits up until the qth set bit is the left partition and the remaining bits define the right. For instance, assuming $q = 1$ and $d = 8$, the d–bit symbol 00101010 has 001 as its left partition and the remaining 01010 as the right partition. It is possible that the right partition may become empty if the qth set bit appears at the right most position or there are less than q set bits in the input symbol, e.g. $split(00000001) \rightarrow \langle 00000001, \emptyset \rangle$, $split(00000000) \rightarrow \langle 00000000, \emptyset \rangle$. Figure 1 sketches a split coding example.

It is important to note that the right partition, after split coding, is a concatenation of non-prefix-free variable-length codes, making it impossible to decode correctly without the codeword boundaries [18]. These boundaries are actually determined by the left partition, as each symbol either has exactly q set bits ending with a 1, or consists of exactly d bits with fewer than q set bits. Therefore, the codeword boundaries in the left partition can be easily identified through a sequential scan. Additionally, this scan can be accelerated by utilizing rank/select dictionaries.

Another key property of split coding is its support for random access. Suppose we want to retrieve the kth symbol on the input. The preceding $(k1)$ occupy the first x bits in the left partition and y bits in the right partition. While it is challenging to determine y due to the missing codeword boundaries in the right partition, detecting x is feasible by tracing the left partition as described earlier.

The equation

$$x + y = (k-1) \cdot d$$

must hold, as the total number of bits per symbol remains unchanged after split coding. Once x is computed, y can be determined. At this point, we know the starting bit positions of the kth symbol in both the left and right partitions. To extract the left partition, we simply read the bits until either q set bits are encountered or d bits have been read. The information from the left partition then reveals how many bits to read from the right partition. Finally, by concatenating the two partitions and applying the reverse permutation, the target symbol is retrieved.

III. Privacy-Preserving Multimedia Storage, Transfer and Retrieval

Assume Alice uploads her videos and photos to cloud storage and downloads items whenever she wants to access them. Since she is concerned about keeping her repository in plain view, which unauthorized people could access, she encrypts all her images and videos before uploading them. This ensures that no one can access the data without the key that only she knows.

However, encrypting data introduces challenges in everyday life. When Alice wants to browse her gallery, every item she selects needs to be decrypted before being viewed, and this decryption must occur on Alice's device, not in the cloud. As a result, real-time browsing of an encrypted repository becomes difficult due to the overhead of decryption, leading to potential delays that can negatively impact the user experience. Therefore, any reduction in the computational cost of security would allow Alice to continue storing her data encrypted rather than in plain text.

We observe that split coding can significantly reduce this computational load, by approximately 75%. The original work [1] demonstrates that, due to the randomization step, the expected length of the left partition is approximately $\approx n \cdot \frac{2q}{d}$ bits, regardless of the input sequence's distribution. In a practical setting, using a simple byte alphabet with $d = 8$ and $q = 1$, the left partition occupies around 25% of the input size. Encrypting only the left partition while leaving the right partition in plain text ensures privacy, as reconstructing the original data from the right partition is difficult due to the absence of codeword boundaries.

This approach results in approximately a 75% reduction in CPU time, as only one-quarter of the input data needs to be encrypted or decrypted. However, the computational cost of performing the split coding must be accounted for when evaluating the overall performance improvement.

Implementing split coding requires numerous bit gather/scatter operations due to the variable-length coding process, which might initially seem computationally intensive. Contrary to this expectation, thanks to SIMD (Single Instruction, Multiple Data) support for bit gather/scatter operations—specifically the `pdep` and `pext` intrinsics [19], [20]—we can process multiple bytes at once. Additionally, by utilizing SIMD for detecting the rightmost and leftmost bit positions, the coding speed reaches around 250 megabytes per second, and the decoding speed exceeds 350 megabytes per second on a single CPU. As a result, the overhead introduced by split coding has only a marginal effect on the overall encryption/decryption performance.

Furthermore, we observe that the randomization step in split coding may not be necessary to guarantee the expected partition sizes for compressed images and videos. The original work does not assume any specific distribution of input data, proving that the randomization step ensures the expected sizes of the partitions regardless of distribution[1]. Since the output of compression algorithms tends to be random, split coding on entropy-encoded image/video files retains these expectations. In practice, we observe that with parameters $d = 8$ and $q = 1$, the left partitions are about one-quarter of the input size.

Therefore, in cases where reducing overhead is critical, the randomization step can be skipped for compressed multimedia files without affecting the expected partition sizes, thus improving the coding/decoding speed.

After applying split coding, the original file is divided into two: the left partition, about one-quarter of the input size, is encrypted, while the right partition, composed of variable-length codes without codeword boundaries, remains unencrypted. Since the codeword boundaries are encoded in the encrypted left partition, properly decoding the right partition alone becomes infeasible.

This approach, demonstrated in Alice's personal case, can be extended to a corporate scenario. For instance, a hospital may store all its patients' medical imaging data on a server. Due to privacy concerns and regulatory requirements, each image/video is encrypted before being sent to the server. The same method can be applied to reduce the encryption workload in such scenarios.

A. Reducing the Load on Video Conferencing Applications

Another important application of video stream encryption occurs in video conferencing and online meeting systems. Typically, each participant is required to encrypt their stream and decrypt streams from other participants in real time. However, as the number of participants increases, the encryption/decryption load can become significant, potentially hindering real-time communication. To address this challenge, modern systems use alternative architectures. Generally, a media server is employed, where each meeting participant communicates with the server using encrypted streams. The media server receives these encrypted streams, decrypts them, combines them into a single stream, and then re-encrypts and shares it with all participants. In this setup, each user only needs to encrypt their stream and decrypt the combined stream from the server, which is much more efficient than performing one encryption and $(k1)$ decryptions in a k–person conference.

However, this solution shifts the computational load to the media server, increasing the cost of online video conferencing. In such systems, using split coding to encode the video stream can reduce the encryption/decryption workload by approximately 75% for both the media server and the clients.

[1]See the proof of Theorem 1 in [1].

IV. Conclusions

We have proposed a method to reduce the encryption/decryption load using a privacy-preserving data representation. Unlike previous approaches to image/video selective encryption, which focus on encrypting specific components such as JPEG coefficients or I-frames in a video stream, split coding's selection process does not interfere with the compression codec. This allows for easier integration with existing image and video compression schemes, both in hardware and software.

Our approach addresses typical scenarios such as saving and retrieving multimedia files from the cloud or participating in online meetings. However, there are many other applications where privacy is crucial and reducing overhead is essential. For instance, space satellite image communications could benefit from split coding, as it provides an efficient way to manage privacy while minimizing computational overhead.

Acknowledgment

This research was, in part, funded by the National Institutes of Health (NIH) Agreement No. 1OT2OD032581. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the NIH."

This research was, in part, also funded by the following NSF grants EPSCoR RII Track4 #2327456, MRI #2320196, and CCF #2315822.

This research was, in part, also supported by the following fellowships: 1) UAF CenterICE Faculty Fellowship and 2) UAF Troth Yeddha Fellowship.

References

[1] M. O. Külekci, "Randomized data partitioning with efficient search, retrieval and privacy-preservation," in *International Computing and Combinatorics Conference*. Springer, 2023, pp. 310–323.

[2] Y. Wang, C. Purev, H. Barndt, H. Toal, J. Kim, L. Underwood, L. Avalo, and A. K. Das, "Toward energy-efficient deep neural networks for forest fire detection in an image," *The Geographical Bulletin*, vol. 64, no. 2, p. 13, 2023.

[3] F. Huettmann, P. Andrews, M. Steiner, A. K. Das, J. Philip, C. Mi, N. Bryans, and B. Barker, "A super sdm (species distribution model)'in the cloud'for better habitat-association inference with a 'big data'application of the great gray owl for alaska," *Scientific Reports*, vol. 14, no. 1, p. 7213, 2024.

[4] W. J. Buchanan, S. Li, and R. Asif, "Lightweight cryptography methods," *Journal of Cyber Security Technology*, vol. 1, no. 3-4, pp. 187–201, 2017.

[5] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 522–533, 2007.

[6] O. A. Khashan and M. AlShaikh, "Edge-based lightweight selective encryption scheme for digital medical images," *Multimedia Tools and Applications*, vol. 79, no. 35, pp. 26 369–26 388, 2020.

[7] M. Abomhara, O. Zakaria, O. O. Khalifa, A. Zaidan, and B. Zaidan, "Enhancing selective encryption for h. 264/avc using advanced encryption standard," *arXiv preprint arXiv:2201.03391*, 2022.

[8] A. Massoudi, F. Lefebvre, C. De Vleeschouwer, B. Macq, and J.-J. Quisquater, "Overview on selective encryption of image and video: challenges and perspectives," *Eurasip Journal on information security*, vol. 2008, no. 1, p. 179290, 2008.

[9] H. Madushan, I. Salam, and J. Alawatugoda, "A review of the nist lightweight cryptography finalists and their fault analyses," *Electronics*, vol. 11, no. 24, p. 4199, 2022.

[10] A. Bar-On, O. Dunkelman, N. Keller, E. Ronen, and A. Shamir, "Improved key recovery attacks on reduced-round aes with practical data and memory complexities," *Journal of Cryptology*, vol. 33, no. 3, pp. 1003–1043, 2020.

[11] S. Sallam and B. D. Beheshti, "A survey on lightweight cryptographic algorithms," in *TENCON 2018-2018 IEEE Region 10 Conference*. IEEE, 2018, pp. 1784–1789.

[12] M. Van Droogenbroeck and R. Benedett, "Techniques for a selective encryption of uncompressed and compressed images," in *advanced concepts for intelligent vision systems (ACIVS)*, 2002.

[13] J.-L. Liu, "Efficient selective encryption for jpeg 2000 images using private initial table," *Pattern Recognition*, vol. 39, no. 8, pp. 1509–1517, 2006.

[14] N. A. Khan, M. Altaf, and F. A. Khan, "Selective encryption of jpeg images with chaotic based novel s-box," *Multimedia Tools and Applications*, vol. 80, no. 6, pp. 9639–9656, 2021.

[15] A. K. Das, M. O. Kulekci, and S. V. Thankachan, "Memory–efficient fm-index construction for reference genomes," in *2022 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2022, pp. 736–739.

[16] H. Toal and A. K. Das, "Variability and trend analysis of a grid-scale solar photovoltaic array above the arctic circle," in *2023 IEEE 24th International Conference on Information Reuse and Integration for Data Science (IRI)*. IEEE, 2023, pp. 242–247.

[17] R. A. Fisher, F. Yates *et al.*, *Statistical tables for biological, agricultural and medical research, edited by ra fisher and f. yates*. Edinburgh: Oliver and Boyd, 1963.

[18] R. B. Muralidhar, "Substitution cipher with nonprefix codes," 2011.

[19] I. Corporation, *Intel Intrinsics Guide*, 2024, accessed: 2024-09-05. [Online]. Available: https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html

[20] H. Amiri and A. Shahbahrami, "Simd programming using intel vector extensions," *Journal of Parallel and Distributed Computing*, vol. 135, pp. 83–100, 2020.