Noname manuscript No.
(will be inserted by the editor)

FPGA Acceleration of Secret Sharing for 3D Data Cubes

Zi-Ming Wu · Tao Liu · Bin Yan✉ · Jeng-Shyang Pan · Hong-Mei Yang

Received: date / Accepted: date

Abstract Secret sharing can protect the security of important secret information in the network environment. However, the computational complexity and processing delay increase drastically when the secret information contains large amounts of data, such as 3D (three-dimensional) data cubes. In order to improve the efficiency of secret sharing on 3D data cubes, this paper proposes a hardware architecture to accelerate the generation of shares and reconstruction of the secret. The proposed hardware architecture parallelizes the secret sharing process and optimizes on this basis to save large amounts of circuit resources. Simulation results show that this architecture performs secret sharing more than ten times faster than the software implementation. This study enables the secret sharing of 3D data cubes, which protect large amounts of data information throughout the process and allows complete reconstruction of 3D data cubes. To be able to handle secrets without decoding them, we then extend the hardware architecture to the four basic processes of multi-party computation, demonstrating the feasibility of the structure and providing preliminary research tools for the effective implementation of multi-party computation.

Keywords Parallel processing · 3D data cubes · Secret sharing · FPGA · Multi-party computation

Bin Yan✉
College of Electronic and Information Engineering, Shandong University of Science and Technology, Qingdao, 266590, Shandong, People's Republic of China
E-mail: yanbinhit@sdust.edu.cn

Zi-Ming Wu
College of Electronic and Information Engineering, Shandong University of Science and Technology, Qingdao, 266590, Shandong, People's Republic of China

1 Introduction

As a carrier of information, 3D data cubes can store more information than 2D (two-dimensional) images and are widely used in various fields, such as medical 3D images, seismic 3D exploration and VR (virtual reality). At the same time, in order to ensure the confidentiality and effectiveness of information in the transmission process, people use cryptography to encrypt the data[1]. But when it comes to situations where the data needs to be handed over to others for processing, there is a risk of revealing secrets. For example, If people's medical records are not encrypted at the time of their visit, some private medical records could be exposed and victims with certain diseases could lose their jobs. People cannot encrypt their information if they want to seek medical treatment. To solve such problems, secret sharing and multi-party computation have been proposed and have been an important area of research for many years[2].

Shamir and Blakley introduced a (t, n)-threshold secret sharing scheme in 1979[3,4,5]. This scheme divides the secret into n shares and distributes them to n users. Each user gets at least one share. Each share alone will not reveal any secret information. Also, these shares are delivered to the users through a private channel, so the users are not aware of other people's information among themselves. When t shares are assembled, the secret reconstruction can be performed by Lagrangian interpolation. This encryption has become an important method for designing secure protocols such as threshold signatures[6], key management[7,8], multi-party computation[9,10,11], electronic voting schemes[12], QR codes[13,14,15] and access control[16]. Since a large amount of computation is required for secret sharing, a limit on the amount of data is needed to

ensure the efficiency of encryption. 2D images contain less information than 3D data cubes, so this encryption method is widely used for 2D information encryption.

Thien and Lin used secret sharing scheme to share 2D secret images for the first time[17]. However, the calculation process is slower and wastes computing memory compared to Boolean sharing. Then, on this basis a series of secret sharing schemes are proposed to deal with different situations, which can be summarized as follows: Boolean operation sharing[18,19], progressive sharing[20,21]and scalable sharing[22]. These schemes are proposed to solve the problem of information security in different situations, but only for information with a small amount of data. Since the byte size of each image pixel is fixed, one puts the secret sharing on a finite field for polynomial computation[23,24]. F. Xing achieved lossy recovery of 2D images by putting the sharing of 2D secret images on a finite field for polynomial operations, and the computational complexity is $O\left(n^3\right)$[25]. C. Qin realized the secret sharing process of 2D images on 2^8 field and realized the complete recovery of 2D images[26]. However, the computational complexity of the multiplication operation on the 2^8 field is higher than that on the 251 field as $O\left(n^4\right)$. 3D data cubes have increased in dimensionality compared to 2D data cubes and require more computation. Compared to CPU, FPGA(Field Programmable Gate Array) can increase computing efficiency by parallel processing. This means that it can process multiple data at the same time and perform different tasks simultaneously. Thus, the processing speed of the whole system is accelerated. So FPGA is considered to increase the speed of sharing.

Stangl J. used FPGA to share files with the different number of storage bytes[27]. The relationship between secret file bit width and sharing efficiency is investigated to provide the possibility of implementing secret sharing of 3D data cubes. Patel uses FPGA to accelerate the MPC process in the public cloud. The bit depth of the subject information in this process is 1[28]. This restricts its usage in sharing images with higher bit depth, such as gray-scale images.

From the above study, secret sharing can encrypt the data without revealing any information and can recover the data losslessly. However, most of the current secret sharing is implemented in software and reduces the computational effort by setting a lower threshold. This results in a low number of generated shares. To address these issues, we use parallel processing of FPGA to accelerate the secret sharing process to generate more shares in less time.

The contribution of this work can be highlighted as follow:

1. Using FPGA to accelerate polynomial multiplication, enabling secret sharing of 3D data cubes.
2. Extending this process to the four basic processes of multi-party computation provides a preliminary research tool for the effective implementation of multi-party computation.

This paper is organized as follows. In section 2, we describe the background knowledge related to our proposal. Section 3 gives the general scheme. Section 4 extends the process to the four basic processes of multi-party computation. Section 5 presents experimental results and discussions. Finally, we conclude this paper in section 6.

2 Related background

In this section, we review existing research and techniques to provide a knowledge base for our work. The basic concepts of 3D data cubes, secret sharing, finite field, and multi-party computation are introduced.

2.1 3D data cubes

3D data cubes can be considered as a stacking of 2D images, and are more substantial in terms of data volume than 2D images.

The main object of this paper is the 3D data cubes. Three slices of a 3D data cubes are shown in Fig. 1. It is assumed that the original 3D data cubes consist of K layers of 8-bit grayscale images of size $M \times W$.

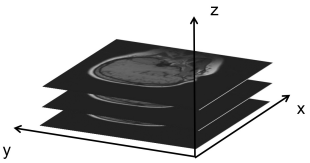


Fig. 1 3D data cubes slices

2.2 Shamir secret sharing

Shamir proposed a (t, n)-threshold scheme by splitting a secret into multiple shares for protection:

(1) By splitting a secret into several copies allows multiple people to work together to manage the recovery of the secret.
(2) When more than t shares can be recovered as the secret, but less than t shares cannot be recovered as the secret.

Secret sharing is divided into a share generation phase and a secret reconstruction phase. All computations are performed in the finite field $GF(p)$, where p

is a prime number which is greater than the maximum value in n and secret.

In the share generation phase, we choose n different elements $x_1, x_2, \ldots,$ and $x_n\,(x_i \geq 0)$ in $GF(p)$ to calculate $f(x_1), f(x_2), \cdots,$ and $f(x_n)$

$$f(x_i) = \left(a_0 + a_1 x_i + \cdots + a_{t-1} x_i^{t-1}\right) \bmod p, \tag{1}$$

where a_0 is the secret and $a_1, a_2, \cdots, a_{t-1}$ are the random numbers uniformly sampled from $GF(p)$. Thus we get n shares, each share and the corresponding x_i is managed by a single user.

In the secret reconstruction phase, one collects any t or more shares and the corresponding x_i. The secret can be reconstructed by Lagrangian interpolation

$$F(x) = \sum_{i=1}^{n} \left(f(x_i) \prod_{\substack{1 \leq \omega \leq n \\ \omega \neq i}} \frac{x - x_\omega}{x_i - x_\omega} \right) \bmod p. \tag{2}$$

When x takes the value of 0, the secret a_0 can be expressed as

$$a_0 = \sum_{i=1}^{n} \left(f(x_i) \prod_{\substack{1 \leq \omega \leq n \\ \omega \neq i}} \frac{-x_\omega}{x_i - x_\omega} \right) \bmod p. \tag{3}$$

2.3 GF finite field

In cryptography, the finite field $GF(p)$ is a very important field, where p is a prime number. Only if p is taken to be prime can all elements of the set be guaranteed to have additive and multiplicative inverse elements.

Since 256 is not a prime number, we can only obtain the largest prime number less than 256 as 251. So people often directly determine the number greater than or equal to 251 as 250. This is often done in image processing, but then the image is not fully recovered[29].

To avoid the loss of pixel data from 251 to 256 introduce $GF(p^n)$. Let p be 2 and n be 8, that is $GF(2^8)$. The calculation characteristics of this field are as follows.

2.3.1 Polynomial addition and subtraction

Suppose there are two polynomials $f(x)$ and $g(x)$

$$f(x) = x^5 + x^4 + x^3 + x \quad and \quad g(x) = x^5 + x^2. \tag{4}$$

The addition of these two polynomials over $GF(2^8)$ can be expressed as

$$\begin{aligned} f(x) + g(x) &= \left(x^5 + x^4 + x^3 + x\right) + \left(x^5 + x^2\right) \\ &= x^4 + x^3 + x^2 + x. \end{aligned} \tag{5}$$

Converting the polynomial in the equation to binary format gives

$$f(x) + g(x) = 00111010 \oplus 00100100 = 00011110. \tag{6}$$

It follows that the addition and subtraction of polynomials are equivalent to the binary XOR (exclusive OR) operation.

2.3.2 Polynomial multiplication

The principle of multiplication is that all numbers in binary can be obtained by dissimilarity with 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80[30]. For example, $x \times 10101101$ can be expressed as

$$x \times (0x80 \oplus 0x20 \oplus 0x08 \oplus 0x04 \oplus 0x01). \tag{7}$$

So we only need to calculate the value of $x \times 0x01, x \times 0x02, \ldots, x \times 0x80$, then all the multiplication results can be obtained.

To calculate multiplication in a finite field, we must first determine an 8-degree irreducible polynomial on $GF(2^8)$. After referring to the AES algorithm, we determine it as

$$x^8 + x^4 + x^3 + x + 1. \tag{8}$$

When the highest bit of the value is 1, the maximum value of the field will be exceeded if the value is shifted further left. So after the shift operation, the value needs to XOR $0x1b$.

The multiplication of these two polynomials on $GF(2^8)$ can be expressed as

$$f(x) \times g(x) = \left(x^5 + x^4 + x^3 + x\right) \times \left(x^5 + x^2\right). \tag{9}$$

Separate $g(x)$ by bit gives

$$f(x) \times g(x) = 0x3a \times 0x24 = 0x3a \times (0x04 \oplus 0x20). \tag{10}$$

Converts a multiplication operation to a shift operation

$$\begin{aligned} &0x3a \times 0x04 = 00111011 << 2 = 11101000, \\ &0x3a \times 0x20 = (00111011 << 5) \oplus 0x1b \\ &\qquad\qquad = 00000001. \end{aligned} \tag{11}$$

The final result of multiplying two polynomials together can be expressed as

$$\begin{aligned} f(x) \times g(x) &= (0x3a \times 0x04) \oplus (0x3a \times 0x20) \\ &= 11101001. \end{aligned} \tag{12}$$

2.4 Multi-party computation

Multi-party computation allows algorithmic computations to be performed directly on the share, and secret reconstruction is performed to obtain the secret for which the same computation has been performed.

This feature is important for protecting the security of information, mainly by taking advantage of the fact that secret sharing is linear encryption. In this process it is possible to protect secret information from being leaked.

2.4.1 Addition process and multiplication process

In the addition process and multiplication process, a constant operation can be added to the shares. The process can be expressed as

$$\begin{aligned} f(x_i) \oplus n &= (a_0 \oplus n) \oplus a_1 x_i \oplus \ldots \oplus a_{t-1} x_i^{t-1}, \\ f(x_i)\, n &= (a_0 n) \oplus (a_1 n)\, x_i \oplus \ldots \oplus (a_{t-1} n)\, x_i^{t-1}. \end{aligned} \tag{13}$$

In these process, n represents the constant, a_0 represents the secret, and $f(x_i)$ represent shares. The reconstruction of the secret can be done by Lagrangian interpolation. The obtained secret will perform the same constant operation.

2.4.2 Double shares addition process

In the double shares addition process, the two shares can be added together and the result of the corresponding secret addition is obtained after decryption. The two shares generation can be expressed as

$$\begin{aligned} f_1(x_i) &= s_j \oplus a_1 x_i \oplus a_2 x_i^2 \oplus \ldots a_{t-1} x_i^{t-1}, \\ f_2(x_i) &= d_j \oplus b_1 x_i \oplus b_2 x_i^2 \oplus \ldots b_{t-1} x_i^{t-1}. \end{aligned} \tag{14}$$

In this equation s_j represents the first secret, d_j represents the second secret, $f_1(x_i)$ represents the first share, $f_2(x_i)$ represents the second share, and the two shares are superimposed additively to obtain

$$f_1(x_i) \oplus f_2(x_i) = (s_j \oplus d_j) \oplus \ldots \oplus (a_{t-1} \oplus b_{t-1})\, x_i^{t-1}. \tag{15}$$

The polynomial generated after the additive superposition remains a polynomial whose number does not exceed $t-1$ times, so it can be directly reconstructed by using Lagrange interpolation. The recovery process is shown in Fig. 2.

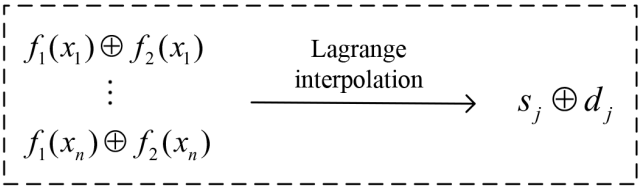


Fig. 2 Double shares addition process

2.4.3 Double shares multiplication process

The double shares multiplication process can be expressed as

$$f_1(x_i)\, f_2(x_i) = (s_j d_j) \oplus \ldots \oplus (a_{t-1} b_{t-1})\, x_i^{2t-2}. \tag{16}$$

After the multiplicative superposition of the two shares, the polynomial generated is at most $2t-2$ times, and the number of times is completely random. So it cannot be recovered directly by Lagrange interpolation. Therefore, it is necessary to generate polynomials with at most $t-1$ times after the second sharing, and then recover them by Lagrange interpolation. The recovery process is shown in Fig. 3.

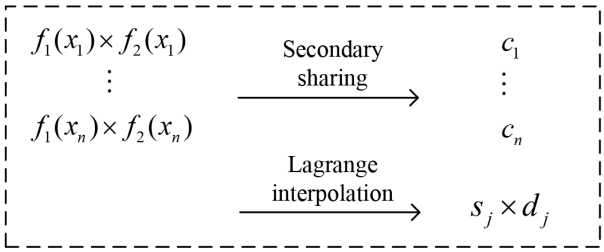


Fig. 3 Double shares multiplication process

Respectively, using the multiplicatively superimposed values $f_1(x_i) \times f_2(x_i)$ as secrets to regenerate the subshares $c_{1\mathrm{i}}, c_{2\mathrm{i}}, \ldots, c_{n\mathrm{i}}$. The subshares are then linearly superimposed by the recombination coefficients to obtain the secondary shares after sharing $c_1, c_2, \ldots, c_n$. This process can be expressed as

$$\begin{cases} c_1 = \alpha_1 c_{11} + \alpha_2 c_{21} + \cdots + \alpha_n c_{n1} \\ c_2 = \alpha_1 c_{12} + \alpha_2 c_{22} + \cdots + \alpha_n c_{n2} \\ \qquad\qquad \vdots \\ c_n = \alpha_1 c_{1n} + \alpha_2 c_{2n} + \cdots + \alpha_n c_{nn}. \end{cases} \tag{17}$$

3 Proposed scheme

Our scheme can be divided into a share generation phase and a secret reconstruction phase. The algorithm is divided into different modules to build parallel circuits, achieving secret sharing of 3D data cubes on the $GF(2^8)$ field.

3.1 Share generation phase

In the share generation phase, the data of the 3D data cubes are first imported into the data processing module for data pre-processing. In this module, all data are arranged as one-dimensional data. After processing the data, they are sequentially imported into the SGM (share generation module) for polynomial computation. Then final shares are generated. This is shown in Fig. 4.

3.1.1 Share generation module

This module performs secret sharing for each data sample in the 3D data cubes sequentially. The secret data are represented by a_0 and a polynomial of order $t-1$ is constructed for each data.The coefficients $a_1, a_2, \cdots, a_{t-1}$ are randomly generated. Thus, each data can generate n shares by setting x_i in advance, where $p(x)$ is set as an irreducible polynomial. The i-th share is

$$f(x_i) = \left(a_0 \oplus a_1 x_i \oplus \cdots \oplus a_{t-1} x_i^{t-1}\right) \bmod p(x). \quad (18)$$

The degree of polynomials and the threshold t are positively correlated. To understand the specific characteristics of Shamir secret sharing, the (3,3)-threshold shares generation formula is established as

$$f\left(x_i\right) = \left(a_0 \oplus a_1 x_i \oplus a_2 x_i^2\right) \bmod p(x). \quad (19)$$

After randomly generating three groups of a_1 and a_2, the share generation equation of 162 as a secret can be expressed as

$$\begin{cases} f\left(x_1\right) = \left(162 \oplus 2x_1 \oplus 4x_1^2\right) \bmod p(x) \\ f\left(x_2\right) = \left(162 \oplus 2x_2 \oplus 4x_2^2\right) \bmod p(x) \\ f\left(x_3\right) = \left(162 \oplus 2x_3 \oplus 4x_3^2\right) \bmod p(x) \end{cases}. \quad (20)$$

Take x_i as 2, 4 and 6 respectively and substitute them into the calculation, the calculation process can be expressed as

$$\begin{cases} 162 \oplus 2 \times 2 \oplus 4 \times 2^2 = 182 \\ 162 \oplus 2 \times 4 \oplus 4 \times 4^2 = 234 \\ 162 \oplus 2 \times 6 \oplus 4 \times 6^2 = 254 \end{cases}. \quad (21)$$

A complete computation process shows that the process of generating the shares is processed in the same way for each secret data. So the parallelism of FPGA can be considered to pipeline this process.

The SGM is mainly a circuit consisting of random number input ROM, x_i input ROM, PMM (polynomial multiplication module), and XOR module.

The random numbers and x_i are stored in advance in ROM. The values can be passed in directly by calling the driver address, which reduces the circuit running time. The efficiency of the calculation is improved by parallelizing the different multiplication processes. When generating a share, x_i is simultaneously passed into different PMM with a_1 and a_2 for computation respectively. However, $a_1 x_i$ and $a_2 x_i^2$ have different computations, and $a_2 x_i^2$ requires a larger computation step. In order to ensure that the computed results reach the XOR module for computation at the same time, the delay module needs to be added. The Fig. 5 shows the circuit diagram when generating a share.

When the processing object is a 3D data cubes, it is difficult to store the corresponding amount of data in ROM because of the huge data of the 3D data cubes. As the threshold continues to increase, there are more and more PMM in the circuit. This wastes a lot of circuit resources.

So it is necessary to find a way to reduce the circuit resources of the PMM. In order to reduce the resources occupation and increase the speed of synthesis, cyclic multiplication is considered to design the circuit. As shown in Fig. 6.

In the SGM. The use of FIFO instead of ROM for data storage enables simultaneous reading and writing. In this way the large data of the 3D data cubes can be processed. The Sel1 FIFO controls MUX1 to pass in different data for calculation. A share generation requires 6 operations. As shown in Fig. 7.

The whole process consists of 6 operations:

(1) Operation1: control MUX1 to pass in random number a_2 through Sel1 FIFO.
(2) Operation2: calculate $a_2 \oplus 0$ in XOR while controlling x_i FIFO pass in x_1.
(3) Operation3: calculate $a_2 x_1$ in PMM while controlling MUX1 pass in a_1.
(4) Operation4: calculate $a_1 \oplus a_2 x_1$ in XOR while controlling x_i FIFO pass in x_1.
(5) Operation5: calculate $a_1 x_1 \oplus a_2 x_1^2$ in PMM while controlling MUX1 pass in a_0.
(6) Operation6: calculate $a_0 \oplus a_1 x_1 \oplus a_2 x_1^2$ in XOR while controlling the MUX2 enable side to pass out the final result.

It can be seen that a large amount of data is repeatedly passed into the PMM when sharing the 3D data cubes. To improve the speed of data sharing, the polynomial multiplication process therefore needs to be processed in parallel.

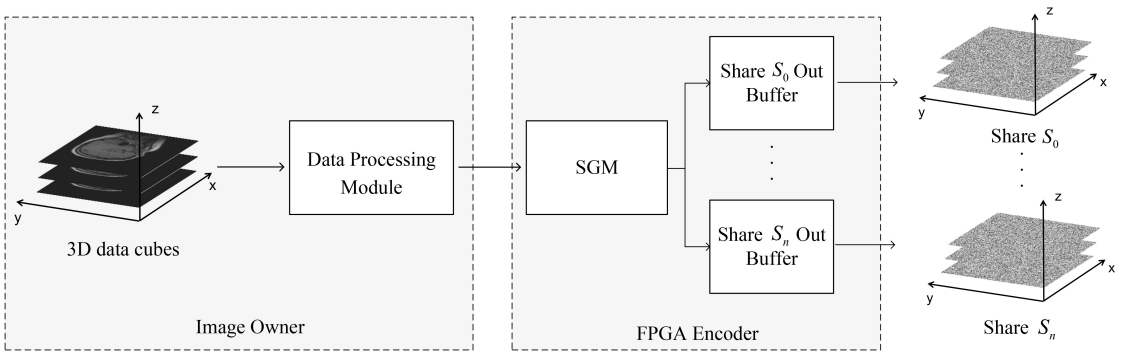


Fig. 4 Share generation phase

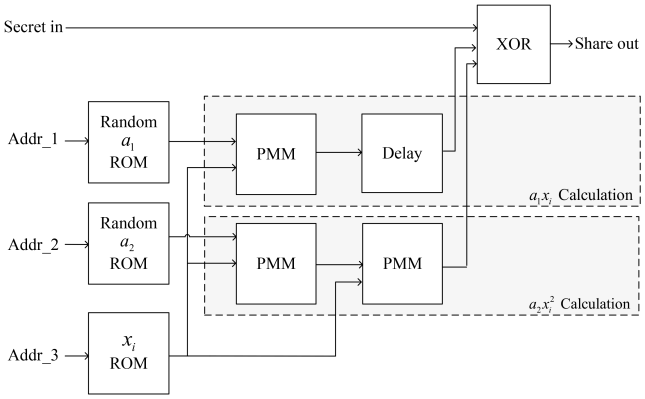


Fig. 5 Structure of the SGM

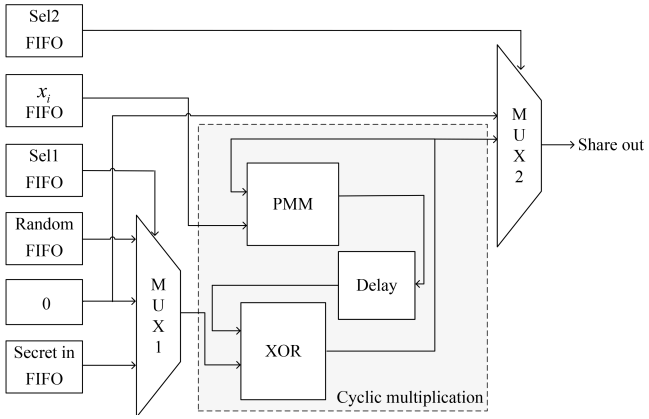


Fig. 6 Optimized SGM

3.1.2 Polynomial multiplication module

In the PMM. First, $X \times 0x01 \ldots X \times 0x80$ should be calculated, and then the non-zero bits of Y should be taken and multiplied with it separately. The two processing processes are carried out in parallel when designing the circuit. We divide it into X processing and Y processing. As shown in Fig. 8 and Fig. 9.

After inputting X, shift X one bit to the right and pass it to XOR. Extract the highest bit of X by Slice to determine if there is a data overflow. If there is data overflow, then you need to control MUX input 0x1b to XOR.

While processing X, the other multiplier Y is shifted one bit to the right. Then, the lowest bit of Y is extracted and multiplied by the processed X. Finally, the results of all multiplications are passed into XOR. The result of multiplying X and Y is obtained.

3.2 Secret reconstruction phase

In the secret reconstruction phase, the original secret data can be restored when t or more shares are collected.

Shares are passed into the SRM (secret reconstruction module) in parallel for Lagrangian interpolation. As a result, the complete original secret is finally obtained. As shown in Fig. 10.

3.2.1 Secret reconstruction module

Lagrangian interpolation method is used directly to perform the recovery. Taking (3,3)-threshold as an example, the polynomial of restoration can be expressed as

$$\begin{aligned} a_0 = & \frac{x_2 \cdot x_3}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \\ & \frac{x_1 \cdot x_3}{(x_2 - x_1)(x_2 - x_3)} f(x_2) + \\ & \frac{x_1 \cdot x_2}{(x_3 - x_1)(x_3 - x_2)} f(x_3) \mod p(x). \end{aligned} \tag{22}$$

The coefficients of the Lagrangian interpolation method are fixed during the secret reconstruction. It can be calculated in advance and passed in through coefficient FIFO to save computation time.

As shown in Fig. 11, the shares and coefficients are passed simultaneously to the PMM through different FIFOs. The terms of the Lagrangian interpolation can be computed at the same time. Finally, the calculated terms are input to XOR to reconstruct the secret.

The designed SGM and SRM can efficiently implement the secret sharing of 3D data cubes. They also

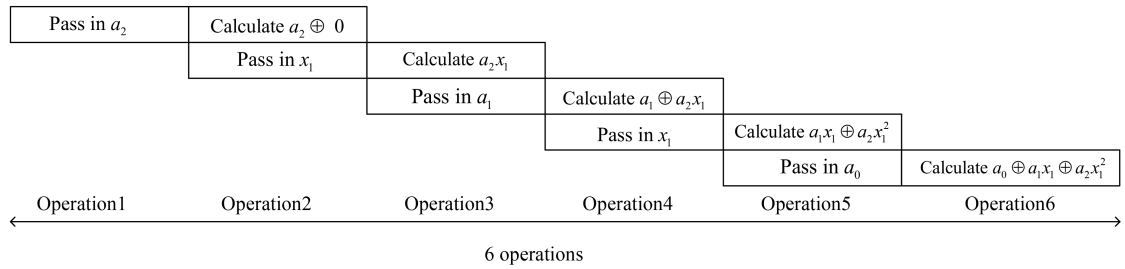


Fig. 7 Loop pipelining

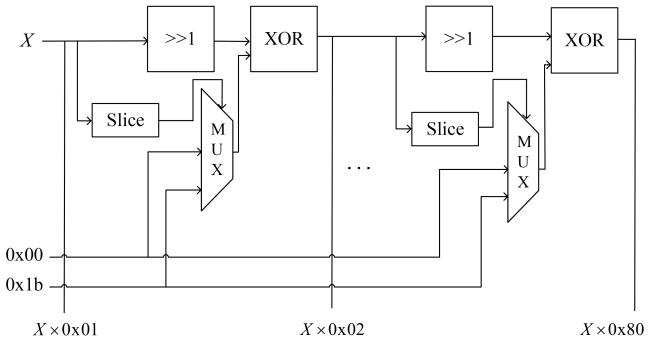


Fig. 8 Polynomial multiplication X processing

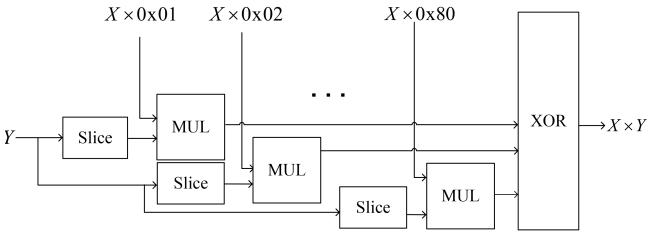


Fig. 9 Polynomial multiplication Y processing

have strong portability. They are also useful when conducting experiments related to secret sharing of 3D data cubes.

4 Extensions to multi-party computation

Our proposed parallel circuits in the previous section can be extended to the four basic processes of multi-party computation, realizing efficient operations for multi-party computation. Section 4.1 describes how to perform operations on shares. Section 4.2 presents an example and analyzes it.

4.1 Four basic processes of multi-party computation

The essence of multi-party computation based on secret sharing is to perform computation on the generated shares. Since the share generation is linear, the reconstructed secret will perform the same computation. Depending on the object of the computation and the method of computation, the basic process of multi-party computation can be divided into additive process, multiplication process, double shares addition process and double shares multiplication process.

As shown in Fig. 12. If the user does not need to encrypt the calculation method, he can select the additive process and the multiplication process. If the user needs to encrypt the calculation method, he can select the double shares addition process and double shares multiplication process.

Algorithm in FIFO passes in the constants to perform operations on the shares. The Sel1 FIFO controls whether the output constants need to be generated as shares as well. The selected constants and shares are then passed into the computing system for operation.

The operations performed in the addition process and double shares addition process are XOR because addition and subtraction are replaced by XOR in the $GF(2^8)$. The computed objects can all be shares or shares and constants.

The operations performed in the multiplication process is multiplication, which can be calculated directly using the ploynomial multiplication module. The calculation is performed on shares and constants. The operation performed in the double shares multiplication process is multiplication. After multiplication, the double shares cannot be reconstructed, so a second sharing is required. The calculation is performed on both shares.

4.2 An example of multi-party computation

We pose a fundamental question to carry out an analysis of how our architecture accelerates multi-party computation.

Suppose Alice and Bob have value1 and value2 respectively. Mike needs to know the sum of value1 and value2, but Mike cannot know what value1 and value2 are.

In this process, the value needs to be shared in order not to leak information. So it is necessary to control MUX1 to select the value passed in after sharing.

As shown in Fig. 13. First, Mike needs to pass the x_i to Bob and Alice. Bob and Alice pass their value

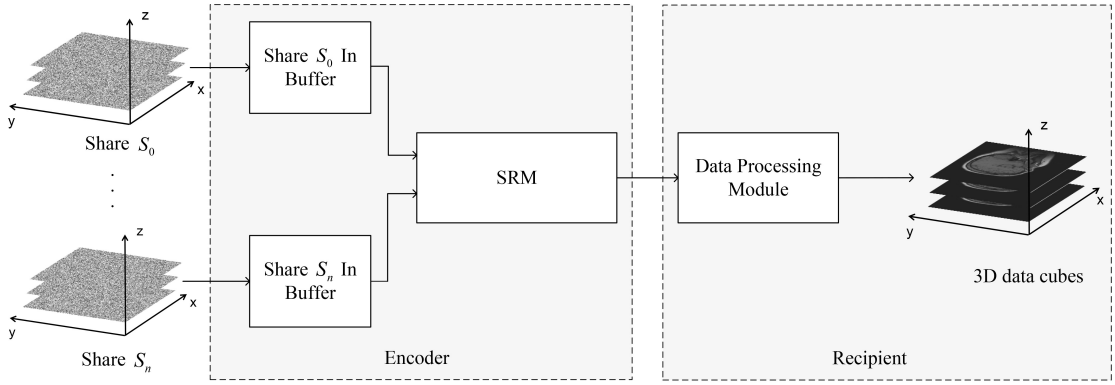


Fig. 10 Secret reconstruction phase

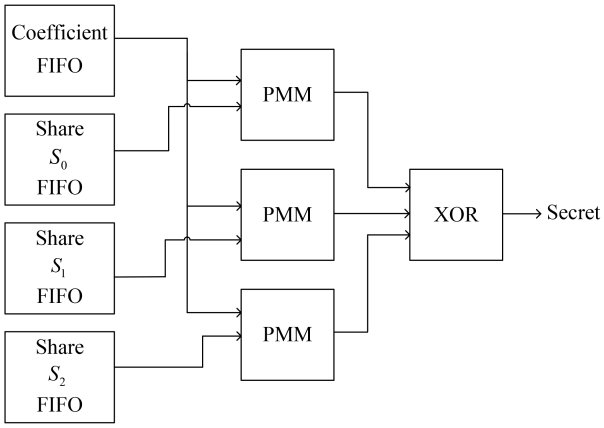


Fig. 11 Structure of the SRM

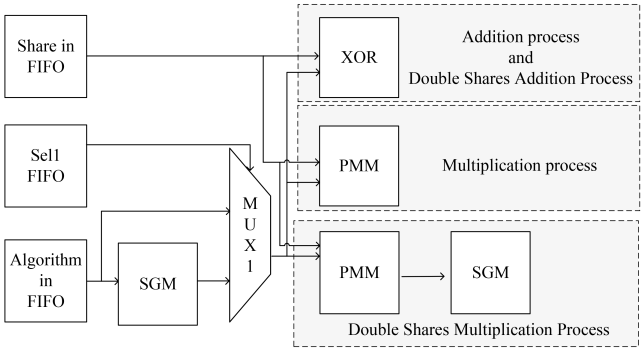


Fig. 12 Multi-party computation

into the SGM to generate the value shares based on the obtained x_i. The value shares are then passed to a trusted third party for computing. After the operation, the value1 shares XOR value2 shares are obtained and passed back to Mike. Finally, Mike passes the received values into the SRM to get value1 XOR value2.

For Bob and Alice, what they pass to the third party is the value after sharing. The third party does not know x_i so it cannot reconstruct value1 and value2. Bob and Alice's values are secure. Mike can only get value1 XOR value2 after performing the reconstruction, so the values of Bob and Alice will not be leaked during the reconstruction process either.

If the algorithm needs to perform multiple operations, multiple secret sharing and secret reconstruction are required. If the amount value is large, such as a 3D data cubes. Serial computation will be very time-consuming compared to parallel. Using our designed structure we can accelerate the process in parallel so that the process can be implemented quickly.

5 Experimental results and discussions

In this section, we use the constructed FPGA system to carry out the secret sharing process of the 3D data cubes. Section 5.1 focuses on the correctness and privacy. The recoverability of the secret images is analyzed by conducting recovery experiments on different numbers of shares. Section 5.2 analyzes the consumption of circuit resources. Section 5.3 focuses on the time complexity. We have made a comparison between serial and parallel runtimes.

We choose different 3D data cubes for the experiments to ensure the accuracy and reliability of the experimental results. In this section, 01-head is used as an example for analysis. Its 3D diagram is drawn with 3D slicer as shown in Fig. 14.

5.1 Correctness and privacy

5.1.1 Recoverability of different numbers of shares

The 3D data cubes can be sliced to get more details for analysis, and it is also a common way to process the 3D data cubes. Since the 3D data cubes have three dimensions, we choose three different slicing directions. These are shown in Fig. 15.

The (3, 4)-threshold is chosen as an example. We use different numbers of shares for secret reconstruction to

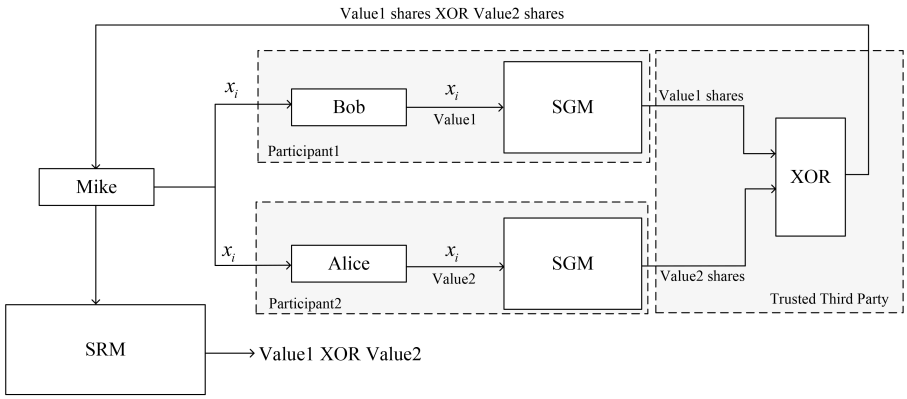


Fig. 13 An example of multi-party computation

Fig. 14 01-head 3D diagram

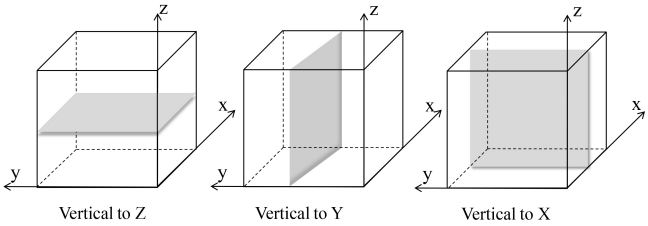


Fig. 15 Slices in three different directions

detect recoverability. The slice with the position in the middle is selected for display. The obtained results are shown in Fig. 16.

We can get it by experiment. When the number of shares is less than 3, the polynomial used to reconstruct the secret will have one item missing. So the Lagrangian interpolation method will not be able to restore the secret. When the number of shares is greater than or equal to 3, the secret image can be recovered. The result is in accordance with the threshold we set.

5.1.2 Entropy calculation for shares

The histogram of an image gives distribution of each value in an image. A natural image usually has a screwed histogram while an encrypted or scrambled image has a flat histogram. So we use histogram of the share to characterize the randomness on share. Fig. 17(a) shows the histogram of the original secret image. The histograms of two shares are shown in Fig. 17(b) and Fig. 17(c), respectively.

As shown in Fig. 17. The histogram of the image before encryption is mainly concentrated around pixel value 20. The pixels of the encrypted image are evenly distributed between 0 and 255. This indicates that the security of the image can be ensured by this scheme.

Entropy in information theory was proposed by Shannon. It can be used to express the degree of redundancy of information.

A signal or source having uniform distribution corresponds to large entropy. So we can use entropy to characterize the randomness of a share image. The entropy can be calculated as

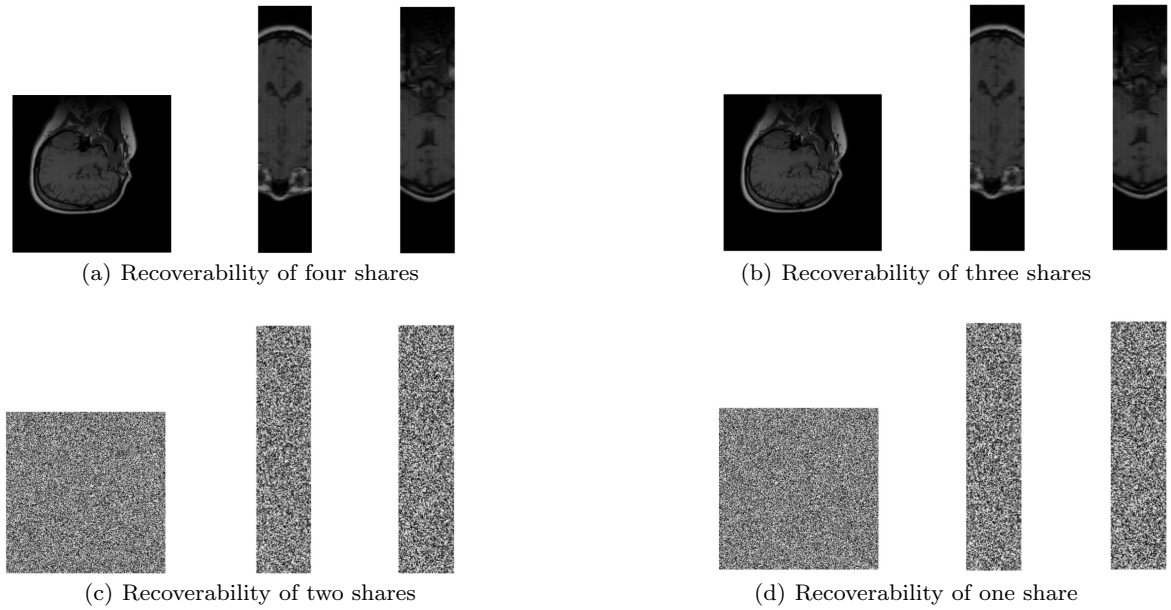
$$H(x_i) = -\sum_{i=1}^{n} p(x_i) \log p(x_i), \tag{23}$$

where n represents the possible values of the random variable, x_i represents the random variable, and $p(x_i)$ represents the probability function of x_i.

As shown in Table 1, the entropy value of the image after sharing is greater compared to the secret image. It means that after the designed system is processed, it has more information security.

5.2 Circuit resource consumption analysis

The circuit resource consumption of FPGA is an important metric for evaluating circuit performance. In this section, a comparison is made between the circuit resource utilization for parallel operation and serial operation. Since the selected comparison scheme is implemented in software and the implementation is serial. In order to be able to compare parallel and serial at the same level, we built circuits for comparison according to the operational characteristics of parallel and serial. Serial can generate only one share at the same time and parallel can generate four shares at the same time. So the parallel implementation requires more circuit resources.



(a) Recoverability of four shares (b) Recoverability of three shares

(c) Recoverability of two shares (d) Recoverability of one share

Fig. 16 The central slice of the secret image recovered from different number of shares

Table 1 Entropy of the histogram

Secret Image	Share at x_i=2	Share image at x_i=4	Share at x_i=6	Share at x_i=8
1.5149	5.1955	5.1949	5.1948	5.1940

Table 2 Circuit resource consumption

(3, 4)-threshold	Serial		Parallel	
	Yan's[25]	Qin's[26]	Optimized circuit	Unoptimized circuit
LUT	56	3053	5018	3270
FF	88	6832	11077	7409
DSP	6	80	105	72

In Table 2, polynomial calculation consumes more circuit resources than normal calculation. In the case of both using polynomial calculation, the serial operation uses 85 DSP (digital signal processing), while the parallel operation uses 105 DSP. The number of LUT (lookup table) is reduced by 35%, the number of FF (flip flop) by 33%, and the number of DSP by 32% after optimization. The optimized circuit has improved the utilization of FF, LUT and DSP.

5.3 Time complexity analysis

In this section, our scheme is compared with Yan's and Qin's. Yan's and Qin's schemes run serially while our scheme runs in parallel. There are two main aspects. Section 5.3.1 is the theoretical analysis, including time complexity and circuit delay. Section 5.3.2 is the actual running time.

5.3.1 Time complexity and circuit delay

The time complexity refers to the number of basic operations in the algorithm, denoted by n as $O(n)$.

Table 3 Computational complexity comparison

(3, 4)-threshold	Algorithm complexity
Yan's[25]	$O(n^3)$
Qin's[26]	$O(n^4)$
Our	$O(n^2)$

Parallel operations use fewer loop structures. As can be seen from Table 3, the time complexity of secret sharing through parallel operations is lower compared to serial operations.

The most significant factor affecting parallel circuits is the circuit delay. It represents the efficiency of the

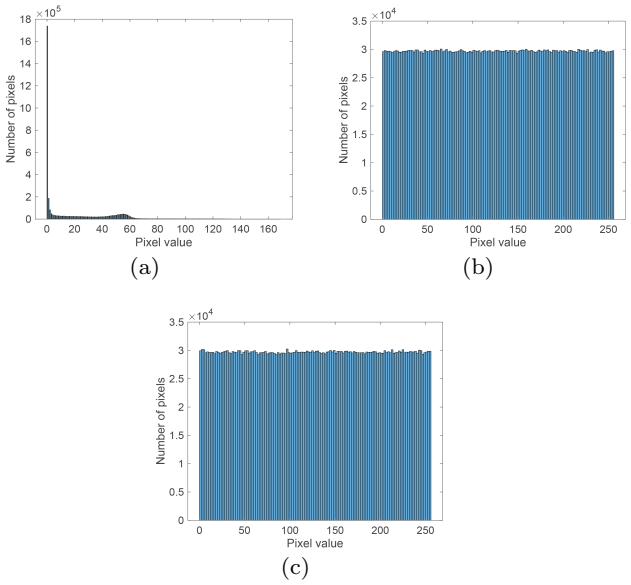


Fig. 17 Original image histogram and share histogram: (a)Secret image pixel histogram. (b)Histogram of share when x_i=2. (c) Histogram of share when x_i=4.

circuit in processing data. So we mainly make a comparison between the circuit delay of serial and parallel processing.

Since the simulation is performed on the system generator, the delay of each device is set based on the base settings of the system generator.

From Table 4 and 5, we can get that our scheme has less circuit delay than Qin's and more than Yan's. This is because the polynomial calculation is more complex, so more circuit structures are required.

As can be seen from the total delay, multiple data can be processed simultaneously during parallel operation. Parallel operation is much more efficient than serial operation when dealing with 3D data cubes.

5.3.2 Actual running time

The actual running time is tested by conducting experiments on different 3D data cubes. The experimental results of some of the 3D data cubes are given as shown in Table 6. As the amount of data in the 3D data cubes becomes larger, the processing efficiency of our solution becomes more and more efficient. The rest of the experimental results are placed on Github. https://github.com/1184745459/3D-data-cubes-experimental-results

As seen in Fig. 18(a), the actual time is linearly related to the circuit delay. This shows that our scheme can indeed accelerate the secret sharing of 3D data cubes. It can be seen in Fig. 18(b), our scheme has a great advantage when dealing with large data volumes of 3D data cubes.

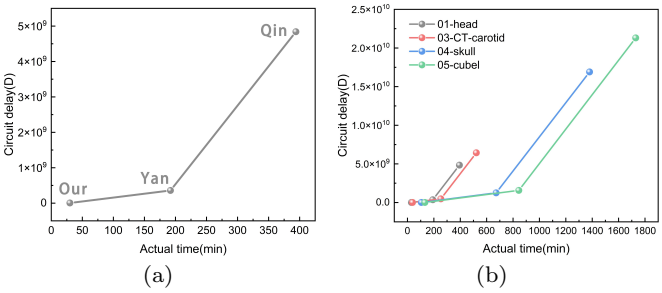


Fig. 18 Actual time and circuit delay:(a)Experimental results of 01-head. (b)Experimental results of different 3D data cubes.

6 Conclusion

In this paper, we apply secret sharing to 3D data cubes without key management while protecting image information. Also, the process is implemented on FPGA using parallelism, which greatly reduces the complexity of computation and improves the efficiency. Extending the process to multi-party computation achieves an efficient implementation of its four basic processes.

Acknowledgements This work was funded by the Shandong Provincial Natural Science Foundation (No. ZR2021MF050), the MOE (Ministry of Education in China) Project of Humanities and Social Sciences (Project No. 18YJAZH110), and the National Statistics Science Project (2021LY082).

Conflict of interest

- Conflict of interest: The authors declare no competing interests.
- Data availability: The data used in this study are available from the corresponding author upon request.
- Authors' contributions: All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by[Tao Liu], [Bin Yan], [Jeng-Shyang Pan] and [Hong-Mei Yang]. The first draft of the manuscript was written by [Zi-Ming Wu] and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

References

1. O. Rolf, Contemporary cryptography. Ph.D. thesis, Artech House, Inc. (2011)
2. O.B. Chanu, A. Neelima, International Journal of Multimedia Information Retrieval **8**(4), 195 (2019). https://doi.org/10.1007/s13735-018-0161-3
3. A. Shamir, Communications of the ACM **22**(11), 612 (1979). https://doi.org/10.1145/359168.359176

Table 4 Circuit delays for different schemes

(3, 4)-threshold	PMM delay	SGM delay	SRM delay	Total delay of the circuit
Yan's[25]	0	44	50	94
Qin's[26]	79	956	239	1274
Our	18	37	55	110

Table 5 Total circuit delay for different schemes

(3, 4)-threshold	Polynomial calculation	Full restoration	Total delay
Yan's[25]	No	No	94× Total number of pixels
Qin's[26]	Yes	Yes	1274× Total number of pixels
Our	Yes	Yes	110+ Total number of pixels

Table 6 Actual time spent in the experiment

(3, 4)-threshold	Number of data	Yan's[25]	Qin's[26]	Our
01-head	3801088	3h12min	6h34min	30min
03-CT-carotid	5023272	4h11min	8h40min	38min
04-skull	13303808	11h12min	22h59min	1h45min
05-cube1	16711680	14h4min	28h52min	2h11min

4. R.J. McEliece, D.V. Sarwate, Communications of the ACM **24**(9), 583 (1981). https://doi.org/10.1145/358746.358762
5. G.R. Blakley, in *International workshop on managing requirements knowledge* (IEEE Computer Society, 1979), pp. 313–313
6. F. Wang, C.C. Chang, L. Harn, Security and Communication Networks **7**(11), 2094 (2014). https://doi.org/10.1002/SEC.921
7. G. Hanaoka, T. Nishioka, Y. Zheng, H. Imai, The Computer Journal **45**(3), 293 (2002). https://doi.org/10.1093/comjnl/45.3.293
8. J. Liu, Y. Wu, X. Liu, Y. Zhang, G. Xue, W. Zhou, S. Yao, The Computer Journal **60**(4), 507 (2017). https://doi.org/10.1093/comjnl/bxw061
9. J. Halpern, V. Teague, in *Proceedings of the thirty-sixth annual ACM symposium on theory of computing* (2004), pp. 623–632
10. H. Pilaram, T. Eghlidos, R. Toluee, IET Information Security **15**(1), 98 (2021). https://doi.org/10.1049/ise2.12007
11. M. Blanton, A. Kang, C. Yuan, in *International conference on applied cryptography and network security* (Springer, 2020), pp. 377–397
12. S. Iftene, Electronic Notes in Theoretical Computer Science **186**, 67 (2007). https://doi.org/10.1016/j.entcs.2007.01.065
13. J.S. Pan, T. Liu, H.M. Yang, B. Yan, S.C. Chu, T. Zhu, Journal of Visual Communication and Image Representation **82**, 103405 (2022)
14. J.S. Pan, X.X. Sun, S.C. Chu, A. Abraham, B. Yan, Engineering Applications of Artificial Intelligence **97**, 104049 (2021). https://doi.org/10.1016/j.engappai.2020.104049
15. T. Liu, B. Yan, J.S. Pan, Applied Sciences **9** (2019). https://doi.org/10.3390/app9214670
16. J. Han, W. Susilo, Y. Mu, J. Yan, The Computer Journal **55**(10), 1202 (2012). https://doi.org/10.1093/comjnl/bxs061
17. C.C. Thien, J.C. Lin, Computers & Graphics **26**(5), 765 (2002). https://doi.org/10.1016/S0097-8493(02)00131-0
18. S. Kabirirad, Z. Eslami, Journal of Information Security and Applications **47**, 16 (2019). https://doi.org/10.1016/j.jisa.2019.03.018
19. S. Kumar, Multimedia Tools and Applications pp. 1–20 (2022). https://doi.org/10.1007/s11042-021-11554-z
20. L. Xiong, X. Han, C.N. Yang, Signal Processing **185**, 108064 (2021). https://doi.org/10.1016/j.sigpro.2021.108064
21. H. Prasetyo, J.W. Simatupang, in *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)* (IEEE, 2019), pp. 1–2
22. R.Z. Wang, C.H. Su, Pattern Recognition Letters **27**(6), 551 (2006). https://doi.org/110.1016/j.patrec.2005.09.021
23. A. Beimel, B. Chor, IEEE Transactions on Information Theory **40**(3), 786 (1994)
24. J.P. Hansen, Contemp. Math **686**, 171 (2017). https://doi.org/10.1090/conm/686/13783
25. F. Xing, X. Yan, L. Yu, Y. Sun, Journal of Visual Communication and Image Representation **86**, 103520 (2022). https://doi.org/10.1016/j.jvcir.2022.103520
26. C. Qin, C. Jiang, Q. Mo, H. Yao, C.C. Chang, IEEE Transactions on Circuits and Systems for Video Technology **32**(4), 1928 (2021). https://doi.org/10.1109/TCSVT.2021.3091319
27. J. Stangl, T. Lorünser, S.M.P. Dinakarrao, in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (IEEE, 2018), pp. 654–659
28. R. Patel, P.F. Wolfe, R. Munafo, M. Varia, M. Herbordt, in *2020 IEEE High Performance Extreme Computing Conference (HPEC)* (IEEE, 2020), pp. 1–8
29. LUOTUO44, (2014). https://blog.csdn.net/luotuo44/article/details/416
30. cptbtptpss, (2014). https://blog.csdn.net/bupt073114/article/details/27