Hindawi
Journal of Electrical and Computer Engineering
Volume 2018, Article ID 3917827, 10 pages
https://doi.org/10.1155/2018/3917827

Research Article

An Efficient Stream Data Processing Model for Multiuser Cryptographic Service

Li Li,[1,2] **Fenghua Li**,[3,4] **Guozhen Shi**,[5] **and Kui Geng**[3]

[1]*College of Communication Engineering, Xidian University, Xi'an 710071, China*
[2]*Department of Electronic and Information Engineering, Beijing Electronics Science and Technology Institute, Beijing 100070, China*
[3]*State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing 100093, China*
[4]*School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China*
[5]*Department of Information Security, Beijing Electronic Science and Technology Institute, Beijing 100070, China*

Correspondence should be addressed to Fenghua Li; lfh@iie.ac.cn

Received 2 April 2018; Accepted 8 July 2018; Published 31 July 2018

Academic Editor: Jar Ferr Yang

In view of the demand for high-concurrency massive data encryption and decryption application services in the security field, this paper proposes a dual-channel pipeline parallel data processing model (DPP) according to the characteristics of cryptographic operations and realized cryptographic operations of cross-data streams with different service requirements in a multiuser environment. By encapsulating cryptographic operation requirements in job packages, the input data flow is divided by the dual-channel mechanism and job packages parallel scheduling, which ensures the synchronization between the processing of the dependent job packages and parallel packages and hides the processing of the independent job package in the processing of the dependent job package. Prototyping experiments prove that this model can realize the correct and rapid processing of multiservice cross-data streams. Increasing the pipeline depth and improving the processing performance in each stage of the pipeline are the key to improving the system performance.

1. Introduction

With the development of computer and network technology, the large number of users and businesses of all kinds of business systems bring huge challenges to data analysis, processing, and storage of business systems. Meanwhile, the urgent need for the security service capabilities of business systems is also put forward. Not only security needs are reflected in financial business, but also the big data analysis for user behavior can easily expose users' personal privacy. The vulnerability of information transmission in the Internet of Things can easily become a security risk in the field of industrial control. The use of cryptographic techniques to ensure the security of business and data and the protection of user privacy are urgent tasks at this stage and even in the future. Therefore, it is necessary to study fast cryptographic operations for mass data. Therefore, considering the security and high-speed processing requirements, it is urgent to design a parallel system that can meet the requirements of different algorithms and different cryptographic working modes.

As the mainstream of computer architecture research and design, multicore has an irreplaceable role in improving computing performance. People have done a lot of research on the high-speed design and implementation of cryptographic algorithm itself, as well as heterogeneous multicore crypto processors. However, there is a lack of research on the high-speed processing of cryptographic services that cross each other in multiuser scenarios. This dissertation takes the design of high-performance cryptographic server as research background. According to the characteristics of cryptographic operations, under the demand of high-concurrency massive data encryption and decryption application service, an efficient stream data processing model for multiuser cryptographic services is proposed to meet the requirements of user-differentiated cryptographic service requirements and achieve high-speed cryptographic service performance.

This paper is organized as follows: Section 2 reviews the existing research. Section 3 introduces the thread separation of the cryptographic operations based on the characteristics of cryptographic operation in different working modes. In Section 4, the dual-channel pipeline parallel data processing model DPP is proposed. Section 5 implements and tests the model.

2. Related Research

As the mainstream of processor architecture development, multicore processors have led to the research upsurge of parallel processing. The speed of data processing is improved by multicore parallel execution. There are two issues involved here: multithread parallelism and multitask parallelism. For multithreaded tasks, concurrent execution of multiple threads by multicore processors can improve the processing performance. For example, one task can be divided into three threads to complete, in the following order: initialization I, operation C, and results output T. Then we can complete it with three cores, as shown in Figure 1. A single-threaded task is usually ported to multiple cores for execution through automatic parallelization techniques. There are many studies on the automatic parallelization of loops. The traditional loop parallel methods include DOALL [1, 2] and DOACROSS [3]. When there is no dependency between iterations of the loop, the DOALL method is used to perform the parallelization sequentially; when there are dependencies between iterations of the loop, the DOACROSS method is used. This automatic parallelization technique cannot achieve good results for general loops containing complex control flow, recursive data structures, and multiple pointer accesses. For this reason, Ottoni proposed an instruction-level automatic task parallel algorithm called Decoupled Software Pipelines (DSWP) [4]. It divides the loop into two parts, the instruction stream and the data stream, and completes parallelism through synchronization between instructions and data. The implementation flow of DSWP and DOACROSS is shown in Figures 2(a) and 2(b), respectively.

DSWP reduce interaction between cores compared to DOACROSS. Whether it is DSWP or DOACROSS, there is a synchronous relationship between threads that are automatically parallelized. For example, the execution time of thread I and thread C needs to be consistent; otherwise, it will result in the waiting consumption of cores. Therefore, synchronization (or coordination) is a key issue that must be considered for parallelization. Concurrency must achieve high performance without significant overhead. Synchronization between threads often leads to the sequentialization of parallel activities, thereby undermining the potential benefits of concurrent execution. Therefore, the effective use of synchronization and coordination is critical to achieving high performance. One way to achieve this goal is speculative execution, which enables concurrent synchronization through thread speculation or branch prediction [5–8]. Successful speculation will reduce the portion of continuous execution, but false speculation will increase revocation and recovery overhead. Simultaneous implementation of the speculative mechanism in the traditional control flow architecture requires a large amount of software and hardware overhead.

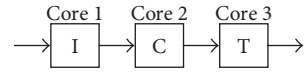

Figure 1: The execution of multithreaded task.

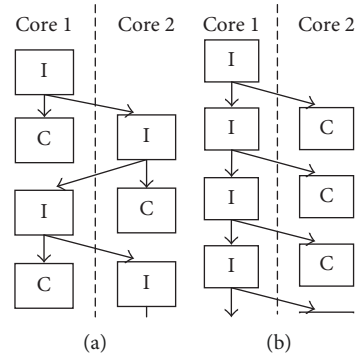

Figure 2: (a) DOACROSS. (b) DSWP.

In multitask parallelism, if each task is a multithreaded task, due to the independence of the tasks and the independence of the threads, the processing method is equivalent to a multithreaded task. If there are single-threaded tasks, after they are parallelized into multiple threads, there will be some associated threads in the process of multitask parallel execution, which must be treated differently.

The cryptographic service involves multiple elements. For example, a block cipher algorithm involves cryptographic algorithms, *KEY*s, initial vectors, working modes, etc. Different cryptographic services have different operational elements. The rapid implementation of multiuser cryptographic services belongs to the multitask parallel domain. As a device for realizing multiuser and massive data cryptographic services, the high-performance cryptographic server must achieve the following two points: First is the correctness of user services: the processing request of different users cannot be confused, and the second is the rapidity of data processing. There are many researches on the fast implementation of the cryptographic algorithm itself, such as improving the computing performance of block cipher algorithm through pipelining [9–13] and optimizing the key operations of public-key cryptography algorithm to improve the operation speed [14–16]. Some studies also accelerate the performance of cryptographic operations through multicore parallelism. For example, the literature uses GPU to implement parallel processing of part of cryptographic algorithms [17–19]. These research results are usually the performance improvement of a single cryptographic algorithm. The literature adopts a heterogeneous multicore approach to complete the parallel processing of multiple cryptographic algorithms [20–22]. However, there is no proposed data processing method for multiuser cryptographic services in the presence of multiple cryptographic algorithms, multiple *KEY*s, and multiple data streams.

This paper proposes a dual-channel pipeline parallel data processing model (DPP), which includes parallel scheduling, algorithm preprocessing, algorithmic operation, and result acquisition. The DPP is designed and implemented in a heterogeneous multicore parallel architecture. This parallel system performs a variety of cryptographic algorithms and supports linear expansion of the algorithm operation unit. Each algorithm operation unit adopts dual channels to receive data and realizes parallel operations among multiple tasks.

3. Thread Split

As mentioned above, different cryptographic services have different computing elements, which is expressed as

$$Service = \{ID, crypto, key, IV, mode\}$$

ID is the service number that is set for multiple users.

The cryptographic operations are usually carried out in blocks and user's data can be divided into several blocks. According to different cryptographic algorithms, the size of the block also varies. Here *UB* is used to represent the size of a single block. The research in this paper is based on the following assumptions:

(1) Parallel processing with *UB* granularity.

(2) Each cryptographic algorithm core completes the operation of one block. The algorithm core adopts the full pipeline design.

The fast implementation of cryptographic algorithm core is not discussed here. Under the premise of meeting the interface conditions, any kind of full pipeline implementation scheme of the cryptographic algorithm can be applied to the algorithm core in this model. This paper focuses on the parallelism between different blocks of cryptographic service.

3.1. Symmetric Cryptographic Algorithm. The parallelism between blocks of symmetric cryptography algorithms must consider the working mode adopted by the cryptographic algorithm. The commonly used working modes are ECB, CBC, CFB, OFB, CTR [23–27], and so on. Assume that C_i denotes the ith ciphertext block, P_i denotes the ith plaintext block, *Enc* denotes the encryption algorithm, *Dec* denotes the decryption algorithm, *key* denotes the *KEY*, *IV* denotes the initial vector, n is the number of plaintext/ciphertext blocks, T_i is the counter value, which increases by 1 with the increment of the block, and u is the length of the last block.

(1) ECB working mode

$$\text{Encryption: } C_i = Enc(key, P_i), \quad 0 \le i < n,$$
$$\text{Decryption: } P_i = Dec(key, C_i), \quad 0 \le i < n.$$

(2) CTR working mode

Encryption:

$$\begin{cases} C_i = P_i \ XOR \ Enc(key, T_i), & i = 0, 1, 2, \ldots, n-2, \\ C_{n-1} = P_{n-1} \ XOR \ MSB_u(Enc(key, T_{n-1})). \end{cases}$$

Decryption:

$$\begin{cases} C_i = P_i \ XOR \ Enc(key, T_i), & i = 0, 1, 2, \ldots, n-2, \\ C_{n-1} = P_{n-1} \ XOR \ MSB_u(Enc(key, T_{n-1})). \end{cases}$$

(3) CBC working mode

Encryption:

$$\begin{cases} C_0 = Enc(key, XOR(IV, P_0)), \\ C_i = Enc(key, XOR(C_{i-1}, P_i)), & 0 < i < n. \end{cases}$$

Decryption:

$$\begin{cases} P_0 = Dec(key, C_0) XORIV, \\ P_i = Dec(key, C_i) \ XOR \ C_{i-1}, & 0 < i < n. \end{cases}$$

(4) CFB working mode

Encryption:

$$\begin{cases} C_0 = Enc(key, IV) XOR \ P_0, \\ C_i = Enc(key, C_{i-1}) \ XOR \ P_i, & 0 < i < n. \end{cases}$$

Decryption:

$$\begin{cases} P_0 = Enc(key, IV) \ XOR \ C_0, \\ P_i = Enc(key, C_{i-1}) \ XOR \ C_i, & 0 < i < n. \end{cases}$$

(5) OFB working mode

Encryption:

$$\begin{cases} C_0 = Enc(key, IV) \ XOR \ P_0, \\ C_i = Enc(key, C_{i-1}) \ XOR \ P_i, & 0 < i < n. \end{cases}$$

Decryption:

$$\begin{cases} S_0 = Enc(key, IV), \\ S_i = Enc(key, S_{i-1}), & 0 < i < n, \\ P_i = S_i XORC_i, & 0 \le i < n. \end{cases}$$

Because there is no dependency between blocks in the ECB and CTR modes, blocks can be processed in parallel. So ECB and CTR are parallel modes and are very suitable for parallel processing. CBC, CFB, and OFB modes have certain dependencies among blocks, so CBC, CFB, and OFB are serial modes. When using multicore parallel operations in multiuser scenarios, attention must be paid to coordination and synchronization among blocks.

By analyzing each working mode, we can divide the encryption and decryption operation into 3 threads. Thread 1 completes the acquisition of the algorithm core input data, thread 2 completes the encryption/decryption operation of a single block, and thread 3 completes the output of the ciphertext/plaintext data. Taking CBC encryption mode and OFB decryption mode as examples, the thread splitting is shown in Table 1.

In this way of splitting, the function of thread 2 is relatively simple, which is the cryptographic algorithm operation of one *UB*. Since encryption and decryption operations usually require multiple rounds of confusion and iterative operations, the operation time of thread 2 is longer than that of thread 1 and thread 3. Taking the *SM4* algorithm as an example, each block needs 32 rounds of function operations. In the full pipeline approach, the algorithm architecture is shown in Figure 3, where F0 to F31 represent 32 rounds of

Table 1: Thread split.

CBC encryption mode	OFB decryption mode
Thread 1: $result\ 1 = XOR(X, Pi)$ $X = \begin{cases} IV, & i = 0, \\ C_{i-1}, & 0 < i < n. \end{cases}$	Thread 1: $result\ 1 = \begin{cases} IV, & i = 0, \\ result\ 2_{i-1}, & 0 < i < n. \end{cases}$
Thread 2: $result\ 2 = Enc(Key, result\ 1)$	Thread 2: $result\ 2_i = Enc(Key, result\ 1_i)$
Thread 3: $C_i = result\ 2$	Thread 3: $P_i = result\ 2\ XOR\ C_i \quad 0 \le i < n$

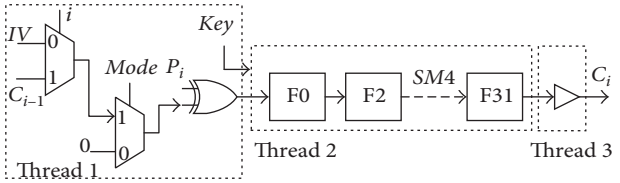

Figure 3: *SM4* 3-thread algorithm operation.

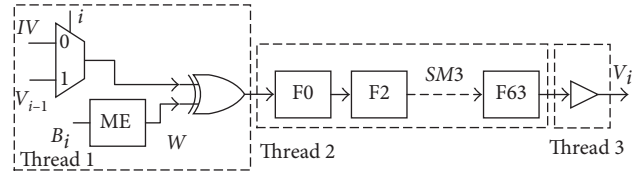

Figure 4: *SM3* 3-thread algorithm operation.

function operations. Different blocks without dependencies can be executed in parallel within it.

For example, *service 1 = {ID1,SM4,key1,IV1,CBC}* and *service 2 = {ID2,SM4,key2,none,ECB}* correspond to the data streams data 1 and data 2, if data 1 can be split into *m UB* blocks and data 2 can be split into *n UB* blocks. That is

$$\begin{aligned} data\ 1 &= \{p_{11}, p_{12}, \ldots, p_{1m}\}, \\ data\ 2 &= \{p_{21}, p_{22}, \ldots, p_{2n}\}, \end{aligned} \tag{1}$$

when p_{1i} is operated in module $F_k\ (0 \le k \le 31)$, $p_{1(i+1)}$ cannot enter the thread 1 module, but p_{2j} can enter the thread 1 and $F_{k'}\ (k' < k)$ modules. So we can insert the block data of the parallel mode between blocks of the serial working mode and hide the execution time of the parallel mode block data inside that of the serial working mode block data. The thread 2 pipeline depth determines the number of independent blocks that can be inserted between dependent blocks.

3.2. Hash Algorithm. The Hash function needs to obtain the hash value of the message through multiple iterations of the block data, so the Hash operation has dependencies between the blocks. By analyzing, Hash algorithm can also be divided into 3 threads: message expansion (ME), iterative compression, and hash value output. The message expansion completes the calculation of parameters required for the iterative compression function, and the hash value output is used for the output of the final result. Taking *SM3* as an example, the algorithm architecture is shown in Figure 4. The parallelism of Hash operations can only occur between blocks of different data streams.

4. DPP Parallel Data Processing Model

The parallel computing models based on PRAM, BSP, and LogP [28, 29] are all computing oriented, lacking pertinence to data processing and are not suitable for practical application of massive data. It can be seen from the above description that the cryptographic operation data in the multiuser environment have the following characteristics: (1) Different user data streams intersect each other. (2) Independent blocks and dependent blocks exist in mutual intersections. Therefore, the parallel processing model must have the following two mechanisms: (1) distinguish between different user data streams and (2) distinguish between data stream dependencies. For this purpose, we encapsulate the data stream and add certain attribute information to the block data, so as to express its properties, thereby facilitating subsequent parallel processing.

The cryptographic operations of streaming data are data-intensive applications. Its typical feature is that the data value is time-sensitive, so the system requires low latency. When a large amount of multiuser data reaches the system in a continuous, fast, time-varying, and cross way, it must be quickly sent to each cryptographic algorithm operation node. Otherwise, data loss may occur due to limited storage space of the system receiver. Referring to the MapReduce data stream processing strategy, specialized module is used to distribute data streams.

The three threads of cryptographic operations of different working modes are implemented by three modules: preprocessing module, operation module, and result output module. The data reorganization module completes the integration of the data stream packages of each service. Figure 5 shows the dual-channel pipeline parallel data processing model proposed in this paper. The data stream processing is divided into six stages: job package encapsulation (PE), parallel scheduling (PS), job package preprocessing (PP), algorithm operation (AO), result output (RO), and data reorganization (DR). The job package encapsulation and data reorganization are completed by the node P_0, the parallel scheduling is completed by the node P_0, and the job package preprocessing, algorithm operation, and result output are completed by the algorithm operation unit *cry_IP*. Each algorithm corresponds to a cluster of algorithm operation units. For example, the operation module cry_IP_{i1} is an encryption operation unit of algorithm s_i and cry_IP_{i2} is a decryption operation unit of algorithm s_i. The dual channel is embodied in the two channels of input and output data of algorithm operation unit.

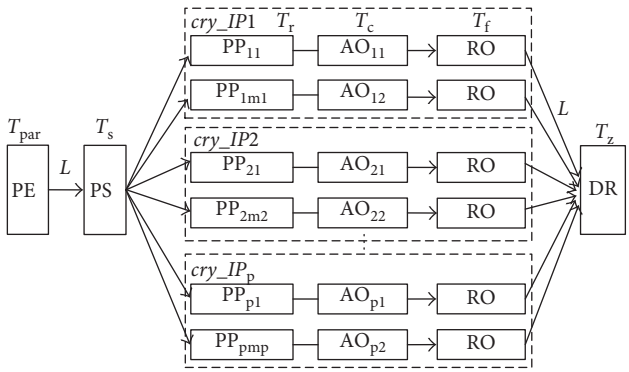

Figure 5: Dual-channel pipeline parallel data processing model (DPP).

In the DPP model, data processing is performed in units of job packages. No explicit synchronization process is required between job packages. Synchronization is implicit in the algorithm preprocessing. Each job package is completed by a fixed algorithm operation unit, and there is no data interaction between the algorithm operation units in the job package processing.

4.1. Encapsulation. The encapsulation is completed by the master node. The format of the encapsulated job package is as follows:

$$P = \{ID, crypto, key, IV, mode, No, flag, l\}$$

ID is the service number set for multiuser and is used to distinguish different data streams; *Crypto* represents the specific demand for cryptographic algorithms and encryption/decryption operations, *key* is the *KEY*, *IV* is the initial vector, and *mode* is the working mode, which can be used to distinguish the dependency property of the job package; No. is the job package serial number, which is used to reassemble the data stream after the algorithm operation. *Flag* is the tail package identifier, which indicates whether the service data flow is end. *l* is the length of the job package and is consistent with the size of the unit block of the cryptographic algorithm.

The flow data received by the system is described as $Task = \{P_{ij}\}$, where *i* corresponds to the service number and *j* corresponds to the sequence number of the service. We refer to job packages in parallel mode as independent job packages and job packages in serial mode as dependent job packages.

4.2. Dual Channel and Parallel Scheduling. According to the thread splitting, under the same algorithm requirements, the difference of package processing of different working modes is embodied in the preprocessing module and the result output module. The operation module does not consider the correlation among the job packages and only processes the input job packages. For this reason, it is necessary to distinguish the input data of the preprocessing module and the result output module. We adopt dual channels to receive job packages and classify independent job packages and dependent job packages.

4.2.1. Dual Receiving Channel of the Preprocessing Module. *Channel 1* is used for the transfer of independent job packages. Considering that the first job package of a data stream in the serial mode is not associated with job packages of other data streams, the job package transmitted in channel 1 satisfies the condition:

$$(mode = ECB|CTR) \quad or \quad (mode = CBC|CFB|OFB \quad and \quad No. = 1)$$

Channel 2 is used for the transfer of dependent job packages. The job package that is transmitted in channel 2 satisfies the condition:

$$(mode = CBC|CFB|OFB) \ and \ No. = /1$$

Suppose that the working mode of four services that request cry_IP_{ab} operation is as follows: *service 1. mode = service 2.mode = CBC*, *service 3.mode = service 4. mode = ECB*, the job packages on channel 1 and channel 2 after parallel scheduling are shown in Figure 6.

The selection of channel is determined by the control signal *Si*, and the default selection is channel 1, that is, $Si = 0$. *cnt* is used to record the execution time of the dependent job package in the algorithm operation unit. When module *cnt* senses that the preprocessing module inputs a dependent job package, the counting starts. It is assumed that the algorithm operation unit needs *m* clock cycles to complete the operation of the dependent job package. When $cnt = m$, the counter clears, $cnt = 0$, sets $Si = 1$ to select channel 2, and inputs the next dependency job package. In other cases, $Si = 0$ and channel 1 is selected. The state flow diagram is shown in Figure 7.

4.2.2. Dual Receiving Channel of the Result Output Module. *Channel 3* is used for the transfer of independent job packages. The job package that is transmitted in channel 3 satisfies the condition:

$$(mode = ECB|CTR) \quad or \quad (mode = CBC|CFB|OFB \quad and \quad flag = 1)$$

Channel 4 is used for the transfer of dependent job packages. The job package that is transmitted in channel 4 satisfies the condition:

$$(mode = CBC|CFB|OFB) \ and \ flag = /n$$

Channel 4 provides support for storing intermediate states in serial mode. Since the result of the tail package does not need to be used as an intermediate state, the tail package's result in serial mode is also output through channel 3.

The choice of channel is determined by the control signal *So*, and channel 3 is the default, that is $So = 0$. The channel selection control signal is the same as that of the preprocessing module. When $cnt = m$, *So* is set to 1 and channel 4 is selected. In other cases, $So = 0$ and channel 3 is selected. When $So = 1$, the job package transmitted by channel 4 has the same *ID* as the job package received by the preprocessing module, as shown in Figure 8.

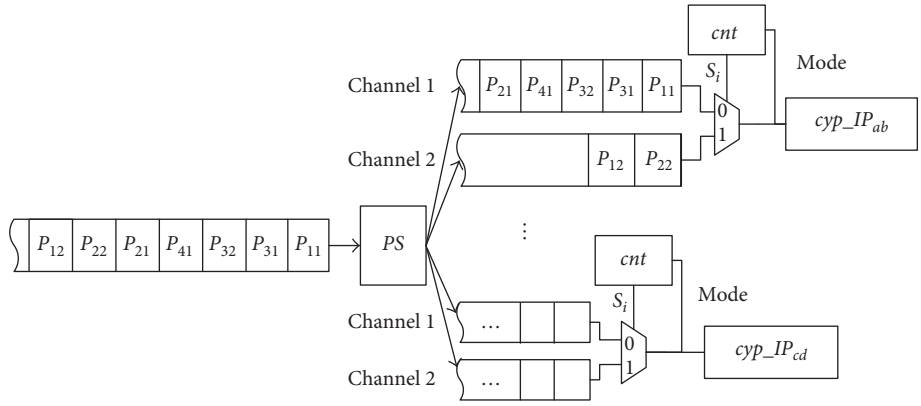

Figure 6: Parallel scheduling and dual receiving channel of preprocessing module.

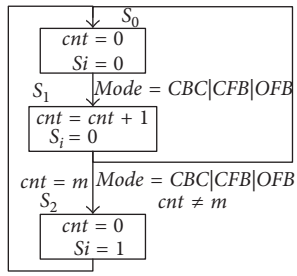

Figure 7: Dual-channel selection control of preprocessing module.

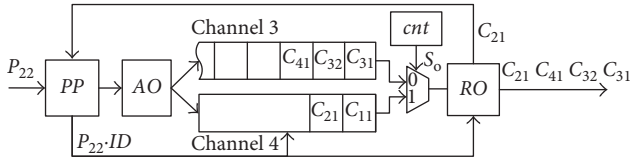

Figure 8: Dual-channel control of output module.

4.2.3. Parallel Scheduling. The process of parallel scheduling is as follows:

Step 1. Determine the algorithm operation unit according to *crypto*.

Step 2. Select the input channel of the preprocessing module of the algorithm operation unit according to *mode* and *No*.

This scheduling method realizes fast transfer of incoming data streams and continuous processing of job packages. The use of dual channels reduces the interaction between modules and hides the processing time of independent job packages in the processing time of dependent job packages, facilitating the parallel execution of job packages.

4.2.4. Data Processing Steps

Step 1. The algorithm application process splits the data to be processed, adds attribute information, and encapsulates it as job package.

Step 2. The algorithm operation unit is determined according to the *crypto* field in the job package, and the input channel of its preprocessing module is selected according to *mode* and *No*. The job package is sent to the corresponding input channel.

Step 3. The preprocessing module obtains the input data of the algorithm operation module, namely, data, *KEY*, and *IV*, according to the package field of *mode* and *No*.

Step 4. The algorithm operation module performs pipeline processing on the received data and sends the result to the receiving channel of the result output module according to *mode* and *flag*.

Step 5. The result output module outputs the received job package to the result receiving process and determines whether to feed the job package back to the preprocessing module according to *mode* and *flag* of the job package.

Step 6. The result receiving process recombines the received job package based on *ID*.

4.3. Parallel Execution Time. Assume that P_0 is the job package encapsulation and reorganization node, and P_a, P_b, and P_c are algorithm operation nodes. T_{par} is the package encapsulation time, and T_z is the data reorganization time. g is the communication interval, that is, the minimum time interval during which node P_0 continuously transmits and receives job packages. The reciprocal of g corresponds to the communication bandwidth. L is the maximum communication delay, which is the time taken to transmit a job package from node P_0 to the scheduling node. T_s indicates the parallel scheduling time of job packages, T_r indicates the job package preprocessing time, T_c indicates the job package operation time, and T_f indicates the job package output time. g represents the calculated load, which is the set of job packages. m is the algorithm operation module pipeline depth. The message delivery process of the job package on DPP is shown in Figure 9.

The continuous sending and receiving of messages needs to meet the conditions:

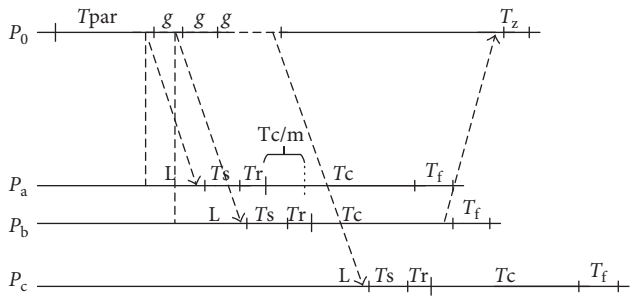

Figure 9: Message delivery on the DPP model.

$$L + T_s + T_r + \frac{T_c}{m} \le g + L + T_s + T_r,$$
$$g \ge \frac{T_c}{m}. \tag{2}$$

Conclusion 1. The system communication bandwidth can be improved by two ways: increasing the operation speed of the algorithm operation module and increasing the pipeline depth of it.

If g job packages come from α data streams, consider two scenarios:

(1) Each data stream adopts a parallel mode; that is, job packages are all independent. The load processing time is as follows:

$$T = T_{par} + (n-1)g + T_s + T_r + T_c + T_f + T_z + 2L. \tag{3}$$

(2) Each data stream adopts a parallel mode, so that job packages of the same service data stream are mutually dependent, and job packages of different service data streams are mutually independent. Assume that the maximum number of service job packages is w', in the extreme case, the first job packet of this data flow appears after other service data flows, and then, the operation time of other service data flows is hidden during that of the longest service data flow. The load processing time is as follows:

$$T = T_{par} + (w - w' - 1)g + w'(T_s + T_r + T_c + T_f) + T_z + 2L. \tag{4}$$

For the data stream mixed in the serial/parallel mode, due to the pipeline design of the algorithm operation module, in the process of dependent job packages, the independent job packages can be executed in parallel, so the execution time of independent job packages is hidden in the execution time of the dependent job package. Therefore, the execution time T of the multitask mixed mode data stream is as follows:

$$T = T_{par} + (n-1)g + T_s + T_r + T_c + T_f + T_z + 2L \le T \le T_{par} + (w - w' - 1)g + w'(T_s + T_r + T_c + T_f) + T_z + 2L. \tag{5}$$

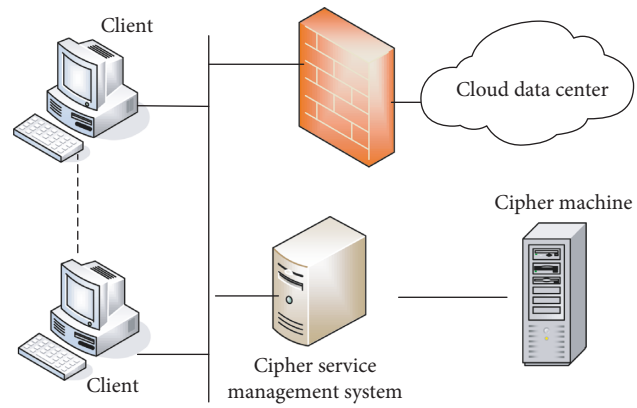

Figure 10: Multiuser cryptographic service.

Conclusion 2. The execution time of mixed cross-data streams is limited to the execution time of the data streams with the most job packages. On the premise of constant pipeline depth, improving the processing performance of each module in the pipeline is the key to improve the processing method.

5. Implementation and Testing

5.1. Hardware Implementation. We prototyped the model to verify its validity. The architecture is shown in Figure 10. The cipher server management system completes the reception of multiuser cryptographic service data streams, the encapsulation of job packages, and the data reorganization service of the operation results. The cryptographic algorithm operation is performed by a crypto machine as a coprocessor. The crypto machine is designed using Xilinx XC7K325t FPGA, which includes parallel scheduling module, *SM3* and *SM4* cryptographic algorithm cores.

The hardware implementation block diagram of the cipher machine is shown in Figure 11. The cipher machine adopts the PCIe interface and receives the job package split by the algorithm application process of the cipher server management system in the way of *DMA* and stores them in *DOWN_FIFO* in the downlink data storage area.

Parallel scheduling module *PSCHEDULE*: Determine the algorithm core according to the *crypto* field of the job package, determine the receiving *FIFO* according to the *mode* and *No.* fields, and realize the transfer of the job package. *FIFO*1 corresponds to channel 1 in the model, and *FIFO*2 corresponds to channel 2 in the model.

Preprocessing module *IP_CTRL*: Acquire algorithm core input data, including *IV*, *KEY*, and the result of the preorder dependent job package, and calculate the input data to cryptographic cores.

Operation modules *SM4* and *SM3*: Cryptographic cores; perform algorithm operations on input data in pipelining way, and send the result of the operation to *uFIFO* or *RAM*. *uFIFO* corresponds to channel 3 in the model, and *RAM* corresponds to the channel 4.

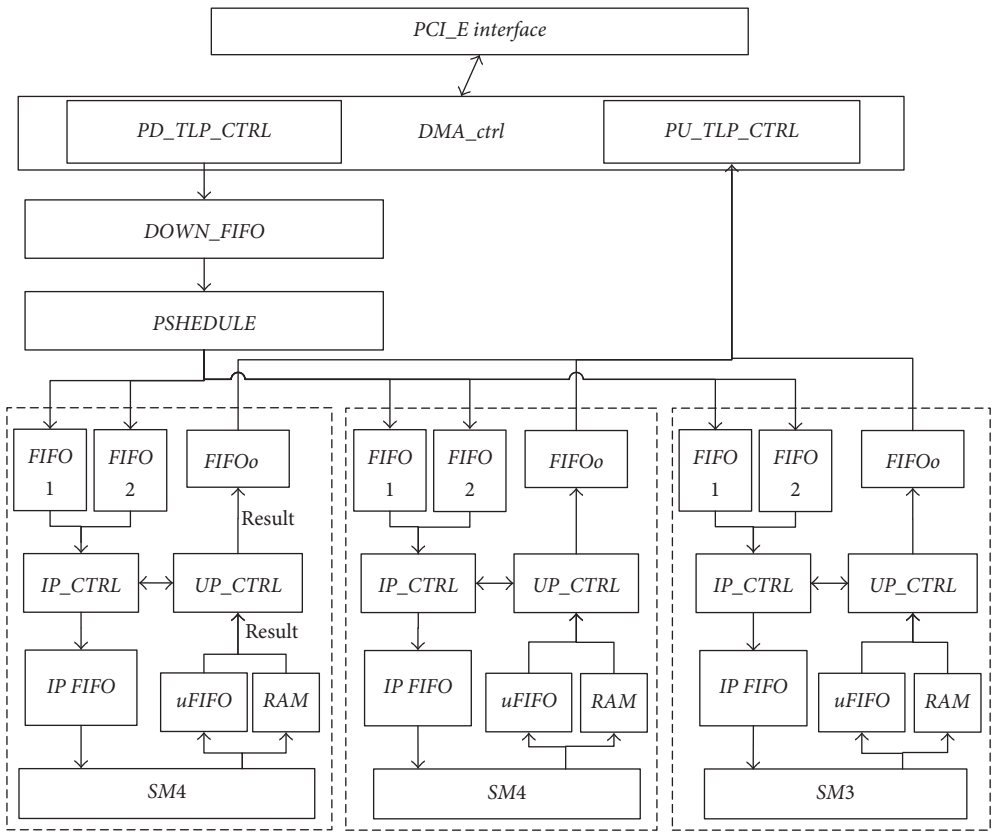

Figure 11: Hardware architecture of partem.

Result output module *UP_CTRL*: If *IP_CTRL* gives the *ID* number, the data of the same *ID* number are extracted from *RAM*, and the output result *result'* is calculated, and fed back to *IP_CTRL* and output to the output *FIFOo* at the same time. If *IP_CTRL* has no *ID* number output, the data in *uFIFO* are extracted, and *result'* is calculated and sent to *FIFOo*. The data in *FIFOo* are fed back to the result reception process of the cipher server management system through the interface module in *DMA* mode.

5.2. Test. The test environment is as follows: The main frequency of the heterogeneous multicore parallel processing system implemented by Xilinx XC7K325t FPGA is 160 MHz, and the interface with the upper application is PCIe 2.0 ∗ 8.

Test 1. SM4 CBC encryption for a 4000 MB file. The end of the test operation takes 114.390935 s, so the data stream processing rate is: 4000 ∗ 8/114.390935 = 279.742446 Mbps.

Test 2. Test 2.1 to Test 2.4 use eight 400 MB files, and job packages of the eight files enter the cipher machine in an interleaving manner. These files use different *IVs* and *KEYs* in different working modes. The end time of each file processing is shown in Table 2. For the data set, the maximum end time is the total time it takes. The data flow processing rate is derived from the following formula:

Table 2: Cipher operation time under cross files.

	File	File 1	File 2	File 3	File 4
	Mode	*SM4 CBC*			
	Time (s)	13.5175	13.4912	13.4994	13.4831
Test 2.1	File	File 5	File 6	File 7	File 8
	Mode	*SM4 CBC*			
	Time (s)	13.4788	13.4872	13.4950	13.5014
	Processing rate: 3200 ∗ 8/13.5175 s = 1.89 Gbps				
	File	File 1	File 2	File 3	File 4
	Mode	*SM4 ECB*			
	Time (s)	12.5029	12.4946	12.4988	12.4784
Test 2.2	File	File 5	File 6	File 7	File 8
	Mode	*SM4 CBC*			
	Time (s)	12.4825	12.4906	12.4864	12.5052
	Processing rate: 3200 ∗ 8/12.5052 s = 2.05 Gbps				
	File	File 1	File 2	File 3	File 4
	Mode	*SM4 ECB*			
	Time (s)	4.3340	4.3166	4.0397	4.1721
Test 2.3	File	File 5	File 6	File 7	File 8
	Mode	*SM4 ECB*			
	Time (s)	4.3433	4.9895	4.1908	4.3623
	Processing rate: 3200 ∗ 8/4.9895 s = 5.13 Gbps				
	File	File 1	File 2	File 3	File 4
	Mode	*SM3*		*SM4 ECB*	
	Time (s)	12.8909	12.4151	13.0394	13.1713
Test 2.4	File	File 5	File 6	File 7	File 8
	Mode	*SM4 CBC*		*SM4 OFB*	
	Time (s)	12.7424	12.6119	13.1909	12.4842
	Processing rate: 3200 ∗ 8/13.1909 = 1.94 Gbps				

$$\text{rate (bps)} = \frac{\text{size of data flow (bit)}}{\text{the total time (s)}}. \quad (6)$$

Analysis: Because Test 1 has only one file in the *CBC* mode, the job packages are interrelated and all are executed serial. Although packages of each file are interrelated, the files of Test 2.1 are independent of each other, so the data flow processing rate of Test 2.1 is higher than that of Test 1. In Test 2.2, 4 files are in the *ECB* work mode, and the independent job packages operation time can be hidden within the operation time of the dependency packages, so the data flow processing rate of Test 2.2 is higher than that of the Test 2.1. Similarly, the data processing rate of Test 2.3 is the highest. Test 2.4 has 2 files with independent job packages and 6 files with dependent packages, but they are allocated in two algorithm units, so the operation rate is close to Test 2.1.

Test 3. Processing rate compare of dual channel and single channel. The total amount of job packages is 10000, and they are randomly assigned to j files. If N_i represents the number of job packages in $file_i$, $\sum_{i=1}^{j} N_i = 10000$. If j is 10, 20, 30, 40, the *ECB* or *CBC* encryption mode is adopted. Change the number of files in *CBC* encryption mode and compare the completion time of data flow in single-channel architecture and dual-channel architecture. The average value of data flow processing time is run several times, and the comparison result is shown in Figure 12. Single 0% means that all files use *ECB* mode, and the system adopts single-channel architecture. Dual 50% indicates that 50% of the files in the data stream use *CBC* mode, and the system is dual-channel architecture, and so on.

As can be seen from Figure 12, when the data flow is an independent data flow, the algorithm operation unit adopts the pipeline design, so the processing rate under the dual channel is close to the processing rate under the single channel; with the increase of the associated job packages in the data flow, the advantage of the data processing rate of dual channel is gradually displayed, and with the increase of the number of files in the data stream, the advantage of the data processing rate is more obvious.

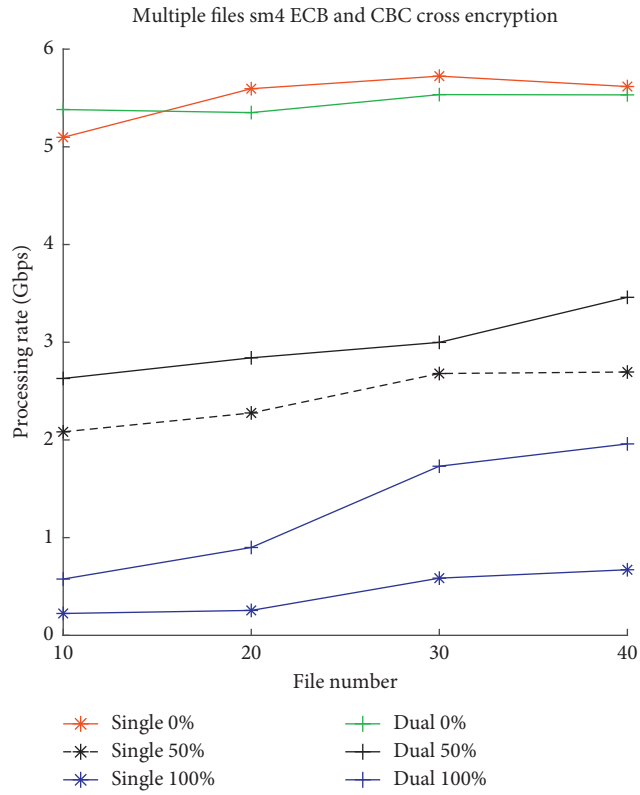

Figure 12: Processing rate comparison of dual channel and single channel.

6. Conclusion

Based on the characteristics of cryptographic operations, this paper proposes a dual-channel pipeline parallel data processing model DPP to implement cryptographic operations for cross-data streams with different service requirements in a multiuser environment. The model ensures synchronization between dependent job packages and parallel processing between independent job packages and data streams. It hides the processing of independent job packages in the process of dependent job packages to improve the processing speed of cross-data streams. Prototype experiments prove that the system under this model can realize correct and rapid processing of multi-service and personalized cross-data streams. Increasing the depth of the cryptographic algorithm pipeline and improving the processing performance of each module in the pipeline can improve the overall performance of the system.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Key R&D Program of China (no. 2017YFB0802705) and the National Natural Science Foundation of China (no. 61672515).

References

[1] Y. Song and Z. Li, "Applying array contraction to a sequence of DOALL loops," in *Proceedings of the International Conference on Parallel Processing (ICPP'04)*, vol. 1, pp. 46–53, Montreal, Canada, August 2004.

[2] G. Elsesser, V. Ngo, S. Bhattacharya, and W. T. Tsai, "Load balancing of DOALL loops in the Perfect Club," in *Proceedings of the 1993 Proceedings Seventh International Parallel Processing Symposium*, pp. 129–133, Newport, CA, USA, April 1993.

[3] D. K. C. Ding-Kai Chen and P. C. Y. Pen-Chung Yew, "Statement re-ordering for DOACROSS loops," in *Proceedings of the 1994 Internatonal Conference on Parallel Processing*, vol. 2, pp. 24–28, Raleigh, NC, USA, August 1994.

[4] G. Ottoni, R. Rangan, A. Stoler, and D. I. August, "Automatic thread extraction with decoupled software pipelining," in *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05)*, pp. 105–118, Barcelona, Spain, November 2005.

[5] V. Krishnan and J. Torrellas, "A chip-multiprocessor architecture with speculative multithreading," *IEEE Transactions on Computers*, vol. 48, no. 9, pp. 866–880, 1999.

[6] A. S. Rajam, L. E. Campostrini, J. M. M. Caamaño, and P. Clauss, "Speculative runtime parallelization of loop nests: towards greater scope and efficiency," in *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, pp. 245–254, Hyderabad, India, May 2015.

[7] S. Aldea, A. Estebanez, D. R. Llanos, and A. Gonzalez-Escribano, "An OpenMP extension that supports thread-level speculation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 78–91, 2016.

[8] J. Salamanca, J. N. Amaral, and G. Araujo, "Evaluating and improving thread-level speculation in hardware transactional memories," in *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 586–595, Chicago, IL, USA, May 2016.

[9] Z. Ying and B. Qinghai, "The scheme for improving the efficiency of block cipher algorithm," in *Proceedings of the 2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, pp. 824–826, Ottawa, ON, Canada, September 2014.

[10] P. Kitsos and A. N. Skodras, "An FPGA implementation and performance evaluation of the seed block cipher," in *Proceedings of the 2011 17th International Conference on Digital Signal Processing (DSP)*, pp. 1–5, Corfu, Greece, July 2011.

[11] L. Bossuet, N. Datta, C. Mancillas-López, and M. Nandi, "ELmD: a pipelineable authenticated encryption and its hardware implementation," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3318–3331, 2016.

[12] P. U. Deshpande and S. A. Bhosale, "AES encryption engines of many core processor arrays on FPGA by using parallel, pipeline and sequential technique," in *Proceedings of the 2015 International Conference on Energy Systems and Applications*, pp. 75–80, Pune, India, October 2015.

[13] T. Kryjak and M. Gorgon, "Pipeline implementation of the 128-bit block cipher CLEFIA in FPGA," in *Proceedings of the 2009 International Conference on Field Programmable Logic and Applications*, pp. 373–378, Prague, Czech Republic, August 2009.

[14] S. Lin, S. He, X. Guo, and D. Guo, "An efficient algorithm for computing modular division over GF(2m) in elliptic curve cryptography," in *Proceedings of the 2017 11th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pp. 179–182, Xiamen, China, 2017.

[15] K. M. John and S. Sabi, "A novel high performance ECC processor architecture with two staged multiplier," in *Proceedings of the 2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE)*, pp. 1–5, Karur, India, April 2017.

[16] M. S. Albahri, M. Benaissa, and Z. U. A. Khan, "Parallel implementation of ECC point multiplication on a homogeneous multi-core microcontroller," in *Proceedings of the 2016 12th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, pp. 386–389, Hefei, China, December 2016.

[17] W. K. Lee, B. M. Goi, R. C. W. Phan, and G. S. Poh, "High speed implementation of symmetric block cipher on GPU," in *Proceedings of the 2014 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pp. 102–107, Kuching, Malaysia, December 2014.

[18] J. Ma, X. Chen, R. Xu, and J. Shi, "Implementation and evaluation of different parallel designs of AES using CUDA," in *Proceedings of the 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)*, pp. 606–614, Shenzhen, China, June 2017.

[19] W. Dai, Y. Doröz, and B. Sunar, "Accelerating NTRU based homomorphic encryption using GPUs," in *Proceedings of the 2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, Waltham, MA, USA, September 2014.

[20] G. Barlas, A. Hassan, and Y. A. Jundi, "An analytical approach to the design of parallel block cipher encryption/decryption: a CPU/GPU case study," in *Proceedings of the 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing*, pp. 247–251, Ayia Napa, Cyprus, February 2011.

[21] H. Kondo, S. Otani, M. Nakajima et al., "Heterogeneous multicore SoC with SiP for secure multimedia applications," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 8, pp. 2251–2259, 2009.

[22] S. Wang, J. Han, Y. Li, Y. Bo, and X. Zeng, "A 920 MHz quad-core cryptography processor accelerating parallel task processing of public-key algorithms," in *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference*, pp. 1–4, San Jose, CA, USA, September 2013.

[23] M. Alfadel, E. S. M. El-Alfy, and K. M. A. Kamal, "Evaluating time and throughput at different modes of operation in AES algorithm," in *Proceedings of the 2017 8th International Conference on Information Technology (ICIT)*, pp. 795–801, Amman, Jordan, May 2017.

[24] A. Abidi, S. Tawbi, C. Guyeux, B. Bouallègue, and M. Machhout, "Summary of topological study of chaotic cbc mode of operation," in *Proceedings of the 2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pp. 436–443, Paris, France, August 2016.

[25] S. Najjar-Ghabel, S. Yousefi, and M. Z. Lighvan, "A high speed implementation counter mode cryptography using hardware parallelism," in *Proceedings of the 2016 Eighth International Conference on Information and Knowledge Technology (IKT)*, pp. 55–60, Hamedan, Iran, September 2016.

[26] H. M. Heys, "Analysis of the statistical cipher feedback mode of block ciphers," *IEEE Transactions on Computers*, vol. 52, no. 1, pp. 77–92, 2003.

[27] M. A. Alomari, K. Samsudin, and A. R. Ramli, "A study on encryption algorithms and modes for disk encryption," in *Proceedings of the 2009 International Conference on Signal Processing Systems*, pp. 793–797, Singapore, 2009.

[28] L. Wang, H. M. Cui, L. Chen, and X. B. Feng, "Research on task parallel programming model," *Journal of Software*, vol. 24, no. 1, pp. 77–90, 2013.

[29] K. Huang, G. C. Fox, and J. J. Dongarra, *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*, Morgan Kaufmann, Burlington, MA, USA, 2011.