GPU Acceleration of Ciphertext-Policy Attribute-Based Encryption

Kai Fan
department of Computer Science
Arkansas State University
State University, USA
kai.fan@smail.astate.edu

Chaoyu Zhang
department of Computer Science
Arkansas State University
State University, USA
chaoyu.zhang@smail.astate.edu

Ruiwen Shan
department of Computer Science
Arkansas State University
State University, USA
ruiwen.shan@smail.astate.edu

Hexuan Yu
department of Computer Science
Arkansas State University
State University, USA
hexuan.yu@smail.astate.edu

Hai Jiang
department of Computer Science
Arkansas State University
State University, USA
hjiang@astate.edu

***Abstract*—With the development of cloud computing, data security became popular in recent decades. However, traditional cryptography has some major limitations. For example, public key cryptography is not scalable in cases with many clients. Since Ciphertext-Policy Attribute-based encryption (CP-ABE) was developed in 2007, it has become as one of the major candidates to implement secure cloud storage. However, CP-ABE still cannot play a solid role due to its several limitations such as complexity of computation, lack efficiency revocation function, etc. This paper will review the CP-ABE and analyze the current CP-ABE toolkit. Major performance bottleneck will be identified and parallelized in CUDA. CP-ABE toolkit will be partially ported to GPU platform for acceleration. Some experiments have been conducted to demonstrate the effectiveness of the proposed approach.**

Keywords—GPU, CP-ABE, Security, Performance, Parallel

I. INTRODUCTION

In recent decades, cloud computing has been widely adopted in the computer world. In cloud centers, a large group of servers is networked so that the computing services or resources can be shared remotely. Cloud computing provides several advantages for its customers, for instance, flexibility, efficiency, and economy, etc. But the primary concern of cloud computing is security.

Cloud computing or cloud storage service allows customers to store their data in remote storage at data centers. Hence, the service provider must ensure their data center infrastructure, as well as their customers' data or applications are secured. Customers who may use a cryptographic system to protect their data can encrypt the data before sending them to servers. However, in traditional public-key cryptography, the sender needs to have the receivers' public key for data encryption. In some cases, we may want to send the same encrypted data to different customers at the same time. Since these people have different public and private key pairs, we have to encrypt the same data with different keys many times based on the number of recipients.

In 2005 Sahai, and Waters [2] initialized the Attribute-Based Encryption (ABE) scheme as a new encryption method with access control. In contrast to traditional public key cryptography, The ABE system does not need to encrypt critical data for a particular user. AES can encrypt critical data for a group which satisfies the. The private keys and ciphertexts of ABE contain a set of attributes or an access control structure. A user can decrypt a ciphertext if and only if his private key attributes math the ciphertext's policy [2].

Based on a different way to assign the policy, ABE has two types, called key-policy ABE (KP-ABE) and ciphertext-policy ABE (CP-ABE). For KP-ABE, during encryption, attributes will be assigned to a ciphertext. The user's private key will be created by an authority, which contains policies. In contrast to KP-ABE, CP-ABE proposed by Bethencourt, Sahai, and Waters in 2007 [1], which support a set of attributes, e.g., roles, and messages can be encrypted with policies defined over a set of attributes. In general, the private key has attributes set, and the ciphertext contains encrypt/decrypt policies. As one of the special schemes of public key cryptography, CP-ABE has three advantages compared to traditional public key cryptography. The first one is only one public key shared by all users. The second advantage is that we usually have one ciphertext, which means that the plaintext and policies are encrypted into one single ciphertext regardless of the number of recipients. As we know, in traditional cases, we need to encrypt the same plaintext multiple times by users' public keys. The third advantage is CP-ABE can provide fine-grain access control much more comfortable than a traditional case. Because of CP-ABE offers expressive rules that define whether the private keys can decrypt ciphertexts [1]. Specifically, the users' private keys have contained a set of

978-1-7281-1651-8/19/$31.00 ©2019 IEEE
SNPD 2019, July 8-11, 2019, Toyama, Japan

attributes or labels, and encryption is associated with an access structure to specify which keys will be able to decrypt

CP-ABE is quite suitable for secure cloud storage. However, there does not exist many CP-ABE implementations in real word due to its complexity. The current CP-ABE toolkit adopts elliptic curve and Cipher Block Chain (CBC) mode AES. Its slowness has prevented it from being deployed in cloud environments. This paper intends to identify the performance bottleneck in CP-ABE toolkit and partially port it to GPU for speedup.

This paper will be arranged as follows: Section 2 will introduce the background of ABE and GPU concepts. Section 3 discusses the related work. Section 4 analyzes CP-ABE toolkit to figure out major components and possible performance issues. GPU version CP-ABE will be proposed in Section 5. Section 6 gives some experimental results. Finally, the conclusion is drawn in Section 7.

II. BACKGROUND

A. Attribute-Based Encryption

ABE is a type of public key encryption which was first introduced by Amit Sahai and Brent Waters [2] as one of the special cases of fuzzy Identity-Based Encryption (IBE). IBE has changed the traditional public key cryptography by using the receiver's identity as the public key to avoid access to a public key certificate [1]. ABE has the same feature, but it is more flexible. The main difference is that ABE uses a set of attributes while IBE uses the receiver's identity as the public key. Compare to IBE and other public key cryptography systems, ABE allows users to have more flexibility and control of the sensitive data.

There are two major forms of ABE cryptography: KP-ABE and CP-ABE. Both of these ideas are inspired by fuzzy IBE, and the only difference between these two methods is the location to save the policy. In CP-ABE, the policy is contained in cipher-texts whereas in KP-ABE, cipher-texts are associated with sets of descriptive attributes.

B. Key-Policy Attribute-Based Encryption

KP-ABE [3] was formulated by Goyal, Sahai, Pandey, and Waters in 2006. In their scheme, the private key in KP-ABE includes the access control tree and defines which cipher-texts can be decrypted by this key. The specific access control structure allows KP-ABE to provide richer types and more flexible cryptosystems than ABE. According to the construction, the definition of KP-ABE was defined by [3] as follows: lease do not revise any of the current designations.

DEFINITION [3]. *Let the following be a set of parties*: $\{P_1 \ldots, P_n\}$. *A collection* $\mathbb{A} \subseteq 2^{\{P_1 \ldots, P_n\}}$ *is monotone if* $\forall B, C: if\, B \in \mathbb{A}\ and\ B \subseteq C\ then\ C \in \mathbb{A}$. *An access structure (resp., monotone access structure) is a collection (resp., monotone collection)* $\mathbb{A}$ *of non-empty sub-sets of* $\{P_1 \ldots, P_n\}$ *i.e.,* $\mathbb{A} \subseteq 2^{\{P_1 \ldots, P_n\}} / \{\emptyset\}$. *The sets in* $\mathbb{A}$ *are called the authorized sets, and the sets not in* $\mathbb{A}$ *are called the unauthorized sets.*

The process of KP-ABE could be divided into four parts [1,3]: Setup, Key Generation, Encryption, and Decryption. Setup randomly generates the public key PK and master key MK. The key generation takes access structure $\mathbb{A}$, MK, and PK as inputs, and generate private decryption key PR as output. During Encryption, the inputs are original data M, a set of attributes γ, and PK, the output of this phase is ciphertext CT. For Decryption, the inputs include the ciphertext CT encrypted with set γ, PR for access control $\mathbb{A}$, and PK, if the attribute set $\gamma \in \mathbb{A}$, the decryption algorithm can decrypt the ciphertext and output the original data M. Otherwise the CT cannot be decrypted by this PR. Fig. 1 illustrates the ciphertexts that labeled with attributes set γ, PR associated with access control structure $\mathbb{A}$ in which each leaf in the access tree is the attribute and each node of the tree indicates the threshold gate. "AND" and "OR" gates are used to present 2 of 2 and 1 of 2 threshold gates respectively.

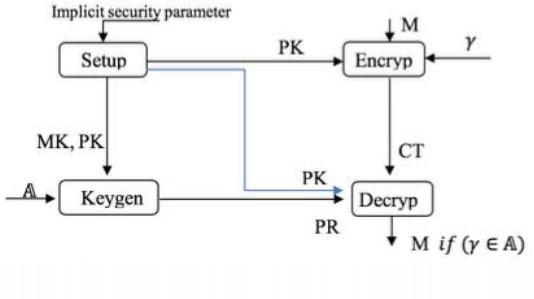

Fig. 1 KEY-POLICY ATTRIBUTE-BASED ENCRYPTION

C. Parallel Computing and GPU Platform

Parallel computing [11] is a kind of computation where a large problem is divided into smaller ones which can be solved concurrently. Since the major manufacturers have started to produce multiple cores in CPU, parallel computing becomes popular. Under multi-cores architecture, each core is independent and can access the same memory at the same time. Within each small piece, the program is executed in sequential order.

Parallel computing is also available on Graphics Processing Unit (GPU) platform specially designed for intensive and highly parallel computations which execute the same instruction with many data elements at the same time. NVIDIA GPU is a typical example. NVIDIA developed Compute Unified Device Architecture (CUDA) in 2006 [4,6], which allows the user to use GPUs for general computing. With CUDA, parallel computing solves many complex computational problems in a more efficient way, the sequential part of the program runs on CPU, while the parallel parts run on thousands of GPU cores simultaneously.

NVIDIA GPU [4,6] consists of a scalable array of multithreaded Streaming Multiprocessors (SMs). When a CUDA program on the host CPU launches a kernel, a grid is created with thread blocks which are the groups of threads executed concurrently on one multiprocessor (MPs) that handle

one or more blocks of a grid. Each MP can be divided into a number of stream processors (SPs) or CUDA cores to host one or more threads of a block.

NVIDIA GPUs provide a hierarchy of memory spaces [4]. Registers and local memory can be read/written by each thread. Shared memory is a small and fast memory within each MP that can be read/write by any thread in a block assigned to that MP. Each grid can read/write the global memory but only can read constant memory and texture memory.

Since thread blocks are executed independently and in any order, CUDA provides the barrier primitive to ensure all threads are synchronized periodically. To avoid unnecessary slowdowns, all these synchronization functions are usually best used for timing purposes or to isolate a failing launch [4].

III. Relate Work

Lewko and Waters [3] proposed a Multi-Authority Attribute-Based Encryption (MAABE) system, also known as decentralizing attribute-Based encryption, in which any party can become an authority to issue private key and generate a public key to different users and there is no requirement for any global coordination. MAABE eliminates the performance bottleneck of central authority and decentralization makes MAABE more scalable than other systems.

Li, and Chen et al., [7] deployed parallel CP-ABE in clouds. In order to reduce the overhead of each algorithm, they used thread-level technology to deploy the parallel program of CP-APE toolkits. The multi-thread strategy led the process of key generation, encryption, and decryption to accelerate 2.28 times to traditional methods. At the last, they built a cloud storage model to illustrate how authentication works in cloud storage.

Shucheng Yu [8] proposed to encrypt data with ABE and store them in untrusted storage. He showed the secure data sharing and access control for cloud computing and wireless sensor networks (WSNs). Muller et al., [9] provided a concept about the distributed Attribute-Based Encryption (DABE), in which the attributes and their corresponding private key can be maintained by any number of parties. Li et al., [10] introduced a method that used ABE to share vital data in cloud computing.

IV. CP-ABE Concepts and Toolkit

This part is mainly about CP-ABE concept and definitions. Then the basic information about the bilinear maps will be introduced. Final part is about current algorithm of CP-ABE, including setup, encrypt, decrypt, and key generation.

A. CP-ABE Concept

CP-ABE was first introduced by Bethencourt et al. [1,13], which provided fine-grained access control. In their design of CP-ABE, a user's private key is associated with a set of attributes and a ciphertext specifies an access policy over a defined universe of attributes within the system. A user will be able to decrypt a ciphertext, if and only if his attributes satisfy the policy of the respective ciphertext [1,13]. For example, in Fig. 2, suppose that a company has several offices over the country. The high-level manager wants to send one sensitive financial statement to offices A and B. This statement has high security degree, which means only the high-level managers like Chief managers and Director of the Financial department in local offices have certain credentials or attributes can read it. The sender may create a special access structure for this information: ((("Chief Managers") OR ("Director") AND ("A" OR "B")) AND (("Financial Department") AND (("Manager level > 5")))). By this mean, the sensitive financial statement should only be seen by Chief Managers or Director of Financial Department in A or B offices, and the priority of managers should be higher than level 5. Thus, the person who encrypts the data doesn't need to know the exact receivers who can read this information, but sender only needs to create an access structure that defines who can access these data.

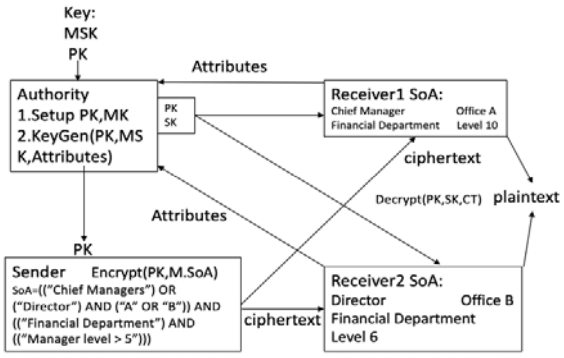

Fig. 2 CP-ABE mechanism

CP-ABE authorization mechanism is different from another traditional public key cryptography. The ciphertext contains the authorization, the receiver's private key satisfies the authorization policy can decrypt the ciphertext. Other public key cryptography systems, the private key, and public key pair must ready before do the encrypt operation, whereas, CP-ABE allows sender encrypts data regardless of whether receivers have their private key. Therefore, the sender can encrypt the plaintext without know the receiver's information. The sender only needs to build a policy over a set of attributes, which define who can access the data. Any future users can decrypt the data only when the user key contains attributes that satisfy the defined policy.

CP-ABE users' private keys will contain an arbitrary number of strings, which expressed as attributes. When we encrypt a message in CP-ABE, the sender only needs to specify the policy. A user who wants to decrypt a ciphertext must satisfy the policy. The policy also calls an access structure. At a mathematical level [1,3,13], access structures in our system are described by a monotonic access tree, where nodes of the access structure are composed of threshold gates, and the leaves describe attributes. We note that AND gates can be constructed as n-of- n threshold gates, and OR gates as 1-of-n threshold gates.

B. CP-ABE Algorithm

Current CP-ABE scheme consists of four fundamental algorithms [1,5]: Setup, Encrypt, Key Generation, and Decrypt.

Setup. The setup algorithm is the first step of CP-ABE, the only input here is the implicit security parameter. The outputs of setup are the public key PK and a master key MK. PK need to be broadcasted to every user and MK is kept by an

authority who maintains the system and manage the key. Authority can use MK to generate the PK.

Key Generation (MK, S). The key generation algorithm using MK and a set of attributes S that describe the key as its input. The output of keygen is a private key SK, which include the user's attributes set.

Encrypt (PK,$M, \mathcal{T}$). The encryption algorithm computes the message M and put the access structure tree $\mathcal{T}$ in ciphertext. It uses a PK, a message M and $\mathcal{T}$ as input, and generate a ciphertext CT that include the $\mathcal{T}$.

Decrypt (PK; CT; SK). The three input for decryption is public key PK, private key PK, and ciphertext CT. The PK include a set of attributes, meanwhile, the CT contains policy policy $\mathbb{A}$. If the attributes of S satisfy the policy $\mathbb{A}$, the CT can be decrypted by this PK, the user will get plaintext M .The decryption algorithm is a recursive function.

C. CP-ABE Toolkit

Bethencourt deployed a CP-ABE toolkit in 2006. It includes two separate packages libbswabe and cpabe. Libbswabe is a library contains the core crypto operations. The cpabe provides high-level functions and user interface. Every package is open source and under the GPL General Public License. The source code is available on the web and can be downloaded from Advanced Crypto Software Collection.

The current implementation uses the Pairing-Based Cryptography (PBC) library which is a free C library and builds on the GMP library. The purpose of the PBC library is to provide a fast and portable library for implementation of pairing-based cryptosystems. It allows a programmer to use their APIs without learning the elliptic curves or other number theory. Therefore, the programmer only needs to have basic knowledge of pairings. However, we are not focused on PBC or any mathematics problems. This paper focus on the procedure of CP-ABE, and how to use GPU parallel computation to accelerate it. The detail of CP-ABE four command line tools will be described in subsections.

1) Cpabe-setup

Cpabe-setup functions are used for generating public parameters and a master key. There is no input for this command except the implicit security parameter. In most situation, the setup function costs very short time and no requests migrate to GPU for speedup.

2) Cpabe-keygen [1,5]

Cpabe-keygen functions' input is MK and S that describe the key. The first step of keygen functions is to call the unserialized functions that unserialize the public key PK and master key MK separately. The most time-consuming part is paring when we run the keygen function. The total execution time depends on the number of attributes and quantity of keys.

3) Cpabe-enc

Cpabe-enc function is the core function for CP-ABE, which includes the encryption and paring two major parts. The inputs of the cpabe-enc function include the public key PK, a message M, and the access structure $\mathcal{T}$ [1,5]. The output of this function is the ciphertext CT, which includes the access structure, also called policy. These functions not only encrypt the plaintext but also need to compute the policies or access structure.

According to Fig. 3, cpabe-enc takes plaintext, public key, and policies as inputs. Once the functions get the public key PK, it will call the pub_unserialize function to unserialize the public key. The policies are taken by a string. The first step is to parse all policies, and then send the policies to simplify() function, which is a recursive function. During this step, the function analyzes the policies based on the "And" or "Or" threshold gate to merge each attribute into the access tree. After the execution of tidy() and format_policy_profix() functions, we get the final policies, which are used as an argument along with unserializing public key and element_t m for function bswabe_enc(). The element_t m is set to the random group element and does not need to be initialized in this function. The bswabe_enc() function uses a random group element for encryption under the specified access policy. The result of this function is returned and the element_t given as an argument is set to the random group element.

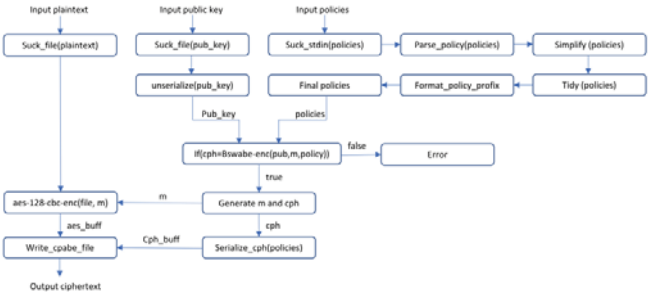

Fig. 3 CPABE-ENC PROCEDURE

After using bswabe_enc() function, it is normal to extract the random data in m using the PBC function, then it is used as the key for AES encryption. The policy is specified as a simple string which encodes a postorder traversal of threshold tree defining the access policy. For example, "tee soda juice 2of3 snacks 1of2", which is specified as a policy with two threshold gates and four leaves. The current version of CP-ABE does support numerical attributes and integer comparison. If there are errors occurred, the function will call bswabe_error(). Otherwise, this function will pass m as a symmetric key for AES encryption, generate policies string cph, serialize the string, and then pass it as an argument for write_back_file() which is used for ciphertext generation.

4) Cpabe-dec [5]

cpabe-dec only takes public key PK, ciphertext CT, and private key SK, which associates with attributes. When the private key SK's attributes satisfy the ciphertext CT policies, this private key can decrypt the corresponding cyphertext and generate plaintext as the output for this function.

Fig. 4 illustrates the procedure of cpabe-dec functions. Once the function gets all inputs, it will do unserialize for all of the inputs, and generate ciphertext string or GbyteArray for *bswabe-dec ()* functions. The *bswabe-dec ()* function decrypt the specified ciphertext using the given private key, filling in the provided element m which does not need to be initialized. If and only if the private key satisfies the ciphertext policy, the *bswabe-*

dec() functions will return true to indicate the success of decryption.

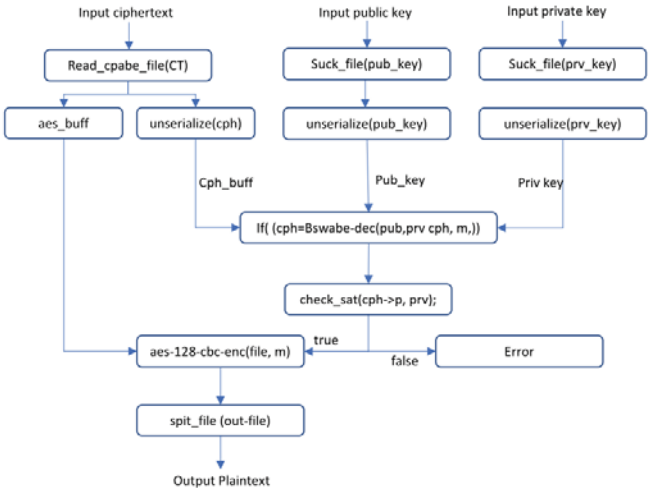

Fig. 4 CPABE-DEC PROCEDURE

After the bswabe-dec () function, aes_128_cbc_enc () takes aes_buf and *m* as arguments, and then conducts AES decryption operation. When we finish the AES decryption, the algorithm will write the file back to the disk. For the cpabe-dec, AES also use CBC mode. Therefore, we will replace it by the CTR mode and try to migrate it to GPU platform for speedup.

D. CP-ABE performance

After introducing the toolkit, we test the original cpabe toolkit with some sample data sets and record the corresponding processing time. The keygen function only need the mater key and attributes set, we run 100 times and generate 100 private keys. The attribute number varies from 1 to 100, which means it is incremented by 1 for each run. The *cpabe-keygen* function highly relies on the PBC library, which exhibits a linear increase by the number of attributes. The testing results are shown in Fig. 5. As we expected, the processing time is perfectly linear with the number of attributes.

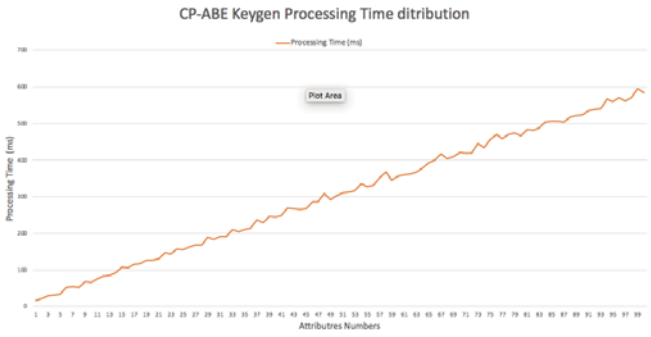

Fig. 5 CPABE-KEYGEN PROCESSING TIME

For *cpabe-enc* and *cpabe-dec*, we divide the processing into steps. As we mention in the previous section, the encryption and decryption procedures are very similar. The process time of reading file, parsing policies, AES encryption/decryption, and writing the file is recorded. We also run encryption and decryption with different sizes of files for 100 times.

Fig. 6 CPABE-ENC PROCESSING TIME DISTRIBUTION

The performance of *cpabe-enc* function is shown in Fig. 6. Since we use a fixed data set, the percentage of encryption execution time varies from 21% to 85%. When the attribute number is incremented by one each time, the processing time for policy will increase, whereas the encryption time remains the same. When the data set is smaller, the policy processing time may dominate the execution time. In contrast, when we use bigger data set the encryption time will to dominate processing time. The time for read and write file operations is quite consistent.

Fig. 7 CPABE-DEC PROCESSING TIME DISTRIBUTION

For *cpabe-dec*, we need to load public key, private key and cpabe ciphertext as well. The loading time for these keys is simple and consistent. The time for loading ciphertext and writing back plaintext is also based on the file size for a linear increase. The policy computation for decryption is shorter than the encryption function because the decryption only needs to check access tree. Whenever we encrypt a file, we need to build an access tree. This needs longer time to compute. From Fig. 7, the decryption time is similar to the encryption time, also based on the file size. The percentage of processing time varies from 62% to 90%.

Therefore, Fig. 6 and 7 show that the processing time breakdown of encryption and decryption functions which dominate the whole processing time.

V. GPU-CPABE

In order to speedup the encryption or decryption process, we adopt NVIDIA CUDA parallel computing platform and take advantage of GPU's thousands of cores for bigger computation power. The first step is to change the CBC mode of AES in the original toolkit to the CTR mode of AES. With the counter mode [12], each block of plaintext is XORed with an encrypted counter that is incremented for each subsequent block. Also, all blocks in CTR mode are totally independent to each other. The CTR mode is popular in high-speed networks encryption

because it can conduct parallelized encryption in both hardware and software levels. Another advantage of CTR mode is simplicity, which means CTR mode only need encryption algorithm, but not the decryption one. The decryption method only needs to replace plaintext to ciphertext.

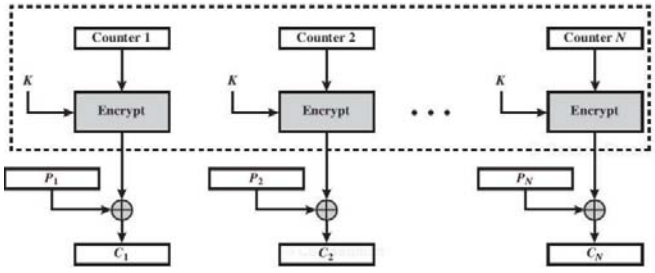

Fig. 8 COUNTER MODE FOR CRYPTOGRAPHIC ALGORITHMS [12]

AES is a block cipher and use symmetric key to encrypt and decrypt messages. AES encryption and decryption are done by each round which is using the same logic. Each round has a different round key that is generated by *key_expand()* function with the original AES key given by the user. So that we can use CPU to compute the key in advance.

The next step we use CPU to compute the round key in advance and store it in GPU memory. For this paper, we chose 128-bit key, means each block is 16 bytes (128 bits). We use a 4×4 matrix for encryption with the look-up table. The AES can be divided into several steps.

1. Initialization: Key expansion is conducted on CPU in advance. Once we get all round keys, the add round key operation is done by XOR'ing the state with the round key.

2. SubByes, Shift Row, and Mix column: Except the first and the last round, other rounds in AES have four separate operations [12]: Substitute bytes, Shift rows, mix columns, and add round key.

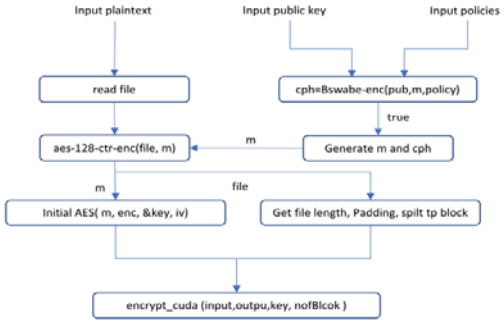

Fig. 9 CUDA CP-ABE Procedure

```
__device__ void encrypt(uint8_t *block, uint8_t *AES_key,
                        uint32_t offset, uint8_t col){
  //add initial round_key
  AddroundKey()
  //run 9 rounds
  for(int i = 1; i < 10; ++i){
      SubBytes();
      shiftRows();
      MixColumns();
      AddroundKey()
  }
  //final round
      subBytes();
      shiftRows();
      AddroundKey();
}
```

Fig. 10 CUDA VERSION OF AES

The CUDA 128 bits CTR AES has 10 rounds in total, once the initial round key is added, the function will run *for* loop nine times and there is no mix columns in the final round.

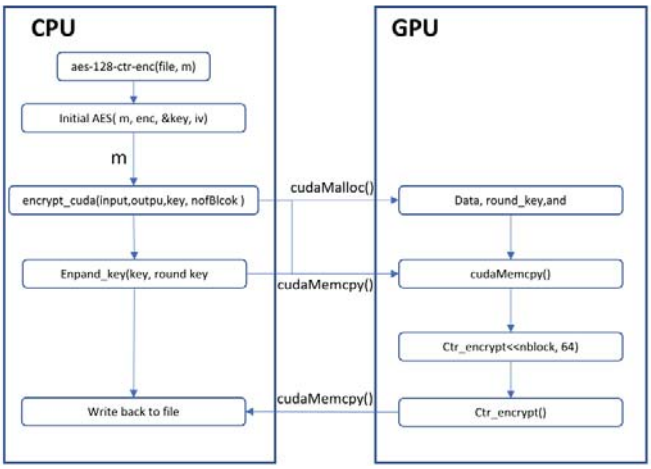

Fig. 11 CUDA Kernel Launch Procerdure

Once AES is ported to GPU, it will accelerate CP-ABE on GPU. In *cpabe-enc* and *cpabe-dec, main()* function calls *aes_128_ctr_encrtypt()* function, and takes the plaintext file pointer and the randomly generated element number as arguments. Once *aes-128-enc()* gets the arguments, it calls *initial AES()* to take *m* and *iv* (initial Vector) as inputs and the second argument in this function indicates if it is an encryption or decryption. Before encryption or decryption, big endianness is set up and *g_byte_arrary_append()* is used to make sure the block number is an exact multiple of 16 for AES data block size of 16 bytes. In CP-ABE we use one and zeroes padding to add bytes of value 0x80 followed by as many zero bytes as necessary to fill up the input for an exact multiple of 16 bytes. Once we get input, key, and the number of blocks, we are ready to run *ecrypt_cuda ()*.

In *encrypt_cuda()* function, the memory is allocated on GPU, and all data need to be copy from CPU to GPU. Then the GPU will launch the kernel and use all threads to compute the data. Once all computations are done, data will be copied back from GPU to CPU and written into files.

```
void encrypt_cuda(const uint8_t *inpuy, uint8_t *output,
                  AES_key *userkey, uint8_t *vi, uint32_t numblock){

  //key expansion run on CPU
  expand_key(userkey, rkey);
  //alocate memory space on device
  cudaMalloc(&dblock, sizeof(uint8_t) * num_bytes);
  cudaMalloc(&drkey, sizeof(uint8_t) * 176);
  cudaMalloc(&dvi, sizeof(uint8_t) * 16);
  //MemcpyHostToDevice
  cudaMemcpy(ddata, (uint32_t *)inparray, sizeof(uint8_t) * num_bytes, cudaMemcpyHostToDevice);
  cudaMemcpy(drkey, (uint32_t *)rkey, sizeof(uint8_t) * 176, cudaMemcpyHostToDevice);
  cudaMemcpy(drvi, (uint32_t *)rseed, sizeof(uint8_t) * 16, cudaMemcpyHostToDevice);

  dim3 nblock((numblock + 32*64 - 1)/(32*64),128);
  //kernel launch
  ctr_encrypt<<<nblock, 64>>>((uint8_t *)ddata, (uint8_t *)drkey, (uint8_t *)drseed, numblock);
  //Synchronize
  cudaThreadSynchronize();
  fflush(stdout);
  //Copy result DeviceToHost
  cudaMemcpy(outarray, ddata, sizeof(uint8_t) * num_bytes, cudaMemcpyDeviceToHost);
  cudaFree(dblock);
}
```

Fig. 12 Code Segment of encrypt cuda()

VI. Experimental Results

We conducted our experiments on a local compute node whose parameters are specified in Table 1. CP-ABE is executed with three data sets (10MB, 100 MB and 500 MB) on CPU and GPU for 100 times respectively. The policy computation is linearly increased and can only run on CPU side. For these three data sets, we did not record the policy execution time as it dominates the whole execution time for 10 MB data. Also, the policy computation highly relies on PBC, which is run on CPU. Therefore, the policy computation part is ignored and the encryption part is considered only.

Table 1 Test Platform Specification

CPU	Intel Core i7 2.3 GHz
Memory	8GB DDR 3
Graphics	NVIDIA GeForce GT 650M 1024MB
Disk	256 SSD
Operating System	Mac OS High Sierra

The time for reading files, writing files, and encryption is recorded. For the selected three data sets, experimental results are listed in Table 2. It is quite clear that bigger data sets will achieve better performance/speedup. For the 500 MB data set, GPU speedup can reach 1.94. The detailed results of all test runs are shown in Fig. 13. The limited performance gains come from our local system resources. For example, we only have 1024 MB on-chip memory for CPU. Also, for big data, if we increase the data set to 1GB, the current algorithm will stop working. Streaming technology has to applied to load data into and out of memory portion by portion.

Table 2 Performance on Three Data Sets

Data	10 MB	100MB	500MB
CPU (ms)	94.49	923.99	4596.19
GPU (ms)	64.57	558.37	2369.01
Speedup	1.46	1.65	1.94

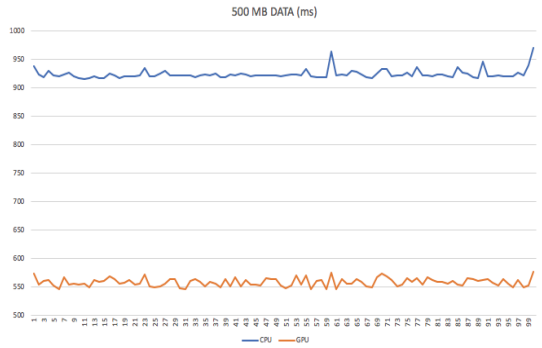

Fig. 13 Performance of encrypt_cuda() on 500 MB Data

VII. Conclusions

This paper focuses on the only available CP-ABE implantation，CP-ABE Toolkit. Its deployment manners and software structure are analyzed thoroughly. Performance bottlenecks are identified and corresponding acceleration strategies are developed. AES encryption and decryption are targeted. A counter mode AES is developed and ported to GPU platform for acceleration. Some experimental results have demonstrated the effectiveness of our approach. The GPU-based CP-ABE can achieve up to 1.9 times speedup on a simple laptop.

Due to the size of two containing libraries in CP-ABE Toolkit, they have not been ported to GPU completely. Therefore, some non-critical components are still executed on CPU. Our future work will port the whole toolkit to GPU for the maximum performance.

VIII. References

[1] A. Sahai J. Bettencourt and B.Waters. Ciphertext-policy attribute based encryp- tion. IEEE Symposium on Security and Privacy, page 321 V334, 2007.

[2] A. Sahai and B. Waters. Fuzzy identity-based encryption. in Proc. EUROCRYPT, page 457473, 2005.

[3] A. Lewko and B. Waters. Decentralizing attribute based encryption. Advances in Cryptology - 30th Annual International Conference on the Theory and Ap- plications of Cryptographic Techniques, 2011.

[4] NVIDIA CUDA Programming Guide 10.0, https://docs.NVIDIA.com/cuda/archive/10.0/cuda-c-programming-guide/index.html.

[5] A. Sahai J. Bethencourt and B. Waters. The cp-abe toolkit. http://acsc.cs.utexas.edu/cpabe/#description.

[6] NVIDIA's KeplerTM GK110/210 Whitepaper, https://images.NVIDIA.com/content/pdf/tesla/whitepaper/kepler- architecture-whitepaper.pdf.

[7] Li, L., Chen, X., Jiang, H., Li, Z., and Li, K.-C., "P-CP-ABE: parallelizing ciphertext-policy attribute-based encryption for clouds", 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 575-580, 2016.

[8] Yu, Shucheng. Data sharing on untrusted storage with attribute-based encryption PhD dissertation, Worcester Polytechnic, 2010.

[9] Müller, S., Katzenbeisser, S., and Eckert, C., "Distributed attribute-based encryption", International Conference on Information Security and Cryptology, pp. 20-36, 2008.

[10] Li, M., Yu, S., Zheng, Y., Ren, K., Lou, W.J.Iou., "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption", IEEE transactions on parallel distributed systems, pp. 131-143. V24, 2013.

[11] Grama, Ananth. Introduction to parallel computing. Pearson Education, 2003.

[12] Stallings, William-Cryptography and network security principles and practice-seventh editon. Pearson, 2017.

[13] Goyal, Vipul, Omkant Pandey, Amit Sahai, and Brent Waters. "Attribute-based encryption for fine-grained access control of encrypted data." In Proceedings of the 13th ACM conference on Computer and communications security, pp. 89-98, 2006.