END-TO-END BIG DATA PROCESSING PROTECTION IN CLOUD ENVIRONMENT USING BLACK BOXES - AN FPGA APPROACH

Lei Xu[1], Khoa Dang Pham[1], Hanyee Kim[2], Weidong Shi[1], and Taeweon Suh[2]
[1] University of Houston, Houston, TX, USA [2] Korea University, Seoul, Korea
{lxu13, pdkhoa, wshi3}@central.uh.edu, {hanyeemy, suhtw}@korea.ac.kr

Abstract

Privacy is one of the critical concerns that hinders the adoption of public cloud. For simple application, like storage, encryption can be used to protect user's data. But for outsourced data processing, i.e., big data processing with MapReduce framework, there is no satisfying solution. Users have to trust the cloud service providers that they will not leak users' data. We propose adding black boxes to the public cloud for critical computation, which are tamper resistant to most adversaries. Specifically, FPGAs are deployed in the public cloud environment as black boxes for privacy preserving computation, and proxy re-encryption is used to support dynamic job scheduling on different FPGAs. In FPGA cloud, cloud is not necessarily fully trusted, and during outsourced computation, user's data is protected by a data encryption key only accessible by trusted FPGA devices. As an important application of cloud computing, we apply FPGA cloud to the popular MapReduce programming model and extend the FPGA based MapReduce pipeline with privacy protection capabilities. Finally, we conduct experiments and evaluation for k-NN with FPGA cloud, which is an important MapReduce application. The experimental results show the practicality of FPGA cloud.

Keywords: Cloud computing, Data security, FPGA, MapReduce

1. INTRODUCTION

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (Mell & Grance, 2009). These features of cloud computing make it attractive for many applications, e.g., database system, customer relationship management, and call center. Especially, as an emerging technology, big data attracts attentions from both the industrial and academic communities.

Security has been considered as one of the critical concerns that hinder the wide adoption of public cloud, especially for the enterprises and the government market. In reality, it is very unlikely that the companies like Amazon, Microsoft, and Google who run the cloud service will try to access the users' data without permission. The main threats come from malicious users and administrators. Due to the virtualization technology used in cloud computing, malicious users may cross the boundary to access others' data (Ristenpart, Tromer, Shacham & Savage, 2009). Administrators usually have higher privileges and they may abuse this ability to learn users' data without permission (like the Snowden case (Toxen, 2014)). For simple cloud service like data storage, the user can protect the data from these threats with encryption. More complex techniques are designed to support applications like sharing and efficient retrieval (Thuraisingham, Khadilkar, Gupta, Kantarcioglu & Khan, 2010). Applications that are dependent on both the storage and computation capability of cloud need a more complex solution for the security concern. FHE (fully homomorphic encryption, (Brakerski & Vaikuntanathan, 2011; Gentry, 2009; van Dijk, Gentry, Halevi & Vaikuntanathan, 2010)) is a powerful tool for privacy preserving computation outsourcing. Because FHE supports operations on cipher-texts, a user only needs to send encrypted data to the cloud for computation. The drawback of FHE based solution is that the existing FHE schemes are very inefficient and it is not practical to use them for meaningful computation tasks such as signal processing and data analysis. Researchers also developed more efficient techniques for special applications such as database operations. There has been work on supporting search on cipher-texts (Boneh, Crescenzo, Ostrovsky & Persiano, 2004; Bellare, Boldyreva & O'Neill, 2007; Song, Wagner & Perrig, 2000), order preserving encryption (OPE) scheme (Boldyreva, Chenette, Lee & O'Neill, 2009; Boldyreva, Chenette & O'Neill, 2011). A cloud database system focusing on managing numerical data was developed based on the OPE scheme (Curino et al., 2011). Hacigümüş, Iyer, Li and Mehrotra (2002) proposed a more general bucketization method to process SQL queries for outsourced database and protect the privacy of the database. Liu, Kantarcioglu, and Thuraisingham (2009) designed a privacy preserving decision tree mining scheme with perturbed data. All these techniques are not fully satisfied because they either support limited application scenarios or suffer from high computation/storage cost. Furthermore, there has been little work in supporting strong privacy preservation for cloud based parallel data analytics such as enabling strong privacy protection for the popular MapReduce programming model.

Field programmable gate arrays (FPGAs) receive much attention in recent years for data analytical applications

because its capability to support customized parallel processing. FPGAs are considered to be powerful computation devices and suited as accelerators for big data analytics. It is common for many applications to employ FPGA based devices to accelerate their performance (e.g., (Court, Gu & Herbordt, 2004; Ronan, Éigeartaigh, Murphy, Scott & Kerins, 2006; Woods & VanCourt, 2008)). FPGAs are also deployed in the cloud environment as accelerators for large scale data processing such as MapReduce (Shan, Wang, Yan, Wang, Xu & Yang, 2010). Another line of FPGA research focuses on protection of the FPGA bitstream (compiled FPGA netlist). In this case, the main concern is to protect the intellectual property of FPGA bitstream developers. Many commercial FPGA devices support bitstream encryption for this purpose. However, to the best of our knowledge, no one has tried to leverage such mechanisms to protect the privacy of the data that is processed by the FPGAs in the public cloud setting.

Since FPGAs are powerful computation devices with certain unique security features that are not available on today's commercial general purpose processors, our motivation is to enable practical privacy preserving solutions in general cloud based data analytics by leveraging the unique security properties of FPGAs. To achieve these goals, we develop a privacy preserving FPGA cloud, a new framework of cloud computation with FPGAs as part of the computation infrastructure provided by a cloud service provider. By taking advantage of the unique security features of FPGAs, one can enable such computing scenario that when privacy sensitive data is processed on the cloud side using FPGAs, the user's data is not disclosed to the service provider who hosts the FPGA cloud infrastructure. As FPGA is a general purpose computation hardware, it enables the user to do many types of computation (e.g., signal processing, data mining, pattern matching, data analytics, and database operations). In a public cloud environment, a user typically doesn't know or care which machine will be assigned to process his/her data. In order to enable the cloud to manage FPGA resources and dynamically allocate arbitrary number of FPGA devices to process user's data, we develop a cryptographic solution based on proxy re-encryption, which supports cipher-text re-encryption. The solution protects cloud user's data key without disclosing it to the proxy. We also discuss how to apply FPGA cloud to MapReduce programming model, which is widely used in the cloud environment for big data processing. In summary, our contributions in this paper include:

- We propose a framework of privacy preserving FPGA cloud, which enables privacy protection in outsourced big data analytics (e.g., MapReduce) by leveraging the existing hardware supports for bitstream IP protection.
- We apply FPGA cloud to MapReduce and extend the FPGA based MapReduce pipeline with privacy protection of user's data against untrusted service providers.
- We design a proxy re-encryption approach and MapReduce key-value transformation that allow cloud service providers to parallelize and manage MapReduce tasks without disclosing user's private data.
- We conduct security and performance analysis, and the results confirm that the FPGA cloud is practical.

The rest of the paper is organized as follows: In Section 2 we give a short review of the techniques used in the proposed solution. Section 3 discusses the security assumptions and threat models. Section 4 provides a detailed description of our solution, followed by security evaluation and enhancement in Section 5. In Section 6, we describe applying FPGA cloud for MapReduce jobs and give a concrete example of k-NN clustering in Section 7. Section 8 discusses related work and Section 9 concludes the paper.

2. BACKGROUND

In this section, we give a brief review of FPGA security features and proxy re-encryption schemes.

2.1 FPGA AND BITSTREAM ENCRYPTION.

FPGAs combine some advantages of software (fast development, low non-recurring engineering costs) with those of hardware (performance, relative power efficiency). These advantages have made FPGAs an important fixture, especially for those applications that are computation intensive. Particularly, a growing body of literature has focused on applying FPGAs to accelerate MapReduce based data analytics (e.g., (Shan et al., 2010)).

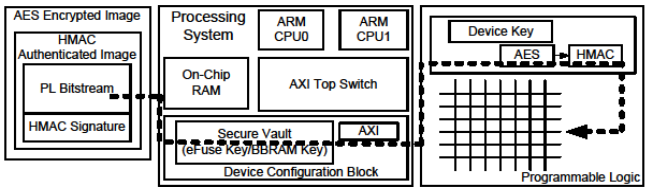


Figure 1. Bitstream encryption for FPGA. The bitstream is protected by AES and HMAC. The keys are also stored in

Most today's FPGAs are configured with bitstreams which completely determine the functionality of the devices. Note that the bitstream is typically stored external to the FPGA in a dedicated configuration memory. Then, it is loaded into the FPGA on every power-up or reset. A disadvantage of this design is that a malicious attacker may learn the bitstream during the loading process, which may result in the compromise of intellectual property. In order to protect valuable intellectual properties of FPGA bitstream developers, many solutions have been proposed to protect the bitstreams themselves. One common approach already implemented by commercially available FPGA devices is to

have a bitstream encrypted and decrypted each time when it is loaded into the FPGA (McNeil, 2012; Microsemi, 2013). Many commercial FPGAs support such bitstream encryption/decryption. Specifically, each FPGA is assigned a symmetric key k and only the legitimate user knows this information. The legitimate user encrypts the bitstream with k and can upload it to FPGA. The encrypted bitstream is often stored in a flash memory that comes with an FPGA based system. After that, the FPGA can decrypt and run the function contained in the bitstream. *Figure 1* depicts the bitstream encryption mechanism. Because the FPGA is tamper resistant, malicious attackers cannot access the bitstream. The symmetric key is protected and cannot be accessed by end users of the FPGA system (e.g., cloud service provider). However, this bitstream IP protection feature cannot be leveraged directly for privacy protection in the cloud. It is unrealistic to distribute the symmetric keys of FPGAs to many different cloud customers securely. To make the matter more complex, users' computation tasks may be dynamically scheduled to different physical FPGA devices in the cloud environment.

2.2 Proxy Re-encryption.

Proxy re-encryption plays an important role in our framework to enable dynamic scheduling. Proxy re-encryption was firstly proposed by Blaze, Bleumer and Strauss (1998), which gave a positive answer to the open problem whether it is possible to convert a cipher-text encrypted using one key to cipher-text encrypted using another key without decryption.

There are usually three parties for a proxy re-encryption scheme, user A, user B, and the proxy. User A and B possess a pair of keys (pk_A, sk_A) and (pk_B, sk_B) respectively. With the key information, a re-encryption key rk_{AB} can be calculated. The re-encryption key is usually sent to the proxy. For any cipher-text c that is encrypted with user A's public key pk_A, the proxy can use rk_{AB} to re-encrypt c to get another cipher-text c'. Then user B can use its own private key sk_B to decrypt the cipher-text c'. During the re-encryption process, the proxy learns nothing about the corresponding plain-text.

Proxy re-encryption schemes with different properties are proposed since the invention of the first scheme (Canetti & Hohenberger, 2007; Libert & Vergnaud, 2008; Tang, 2008). For our purpose, we require that the proxy re-encryption scheme should have the following properties (Ateniese, Fu, Green & Hohenberger, 2006): 1. Unidirectional. Delegation from user A to user B does not allow re-encryption from B to A; 2. Non-interactive. Re-encryption key from user A to B can be generated by A using B's public key, no trusted third party or interaction is required; 3. Collusion safe. Even if a proxy holds a re-encryption key $rk_{A\to B}$ and colludes with user B, they cannot recover user A's private key; 4. Non-transitive. The proxy alone, cannot re-delegate decryption rights, e.g., from re-encryption key $rk_{A\to B}$ and $rk_{B\to C}$, the proxy cannot produce $rk_{A\to C}$. We will explain the importance of these properties for FPGA cloud in Section 5.

3. System Overview and Threat Model

In this section, we give an overview of FPGA cloud and discuss the security assumptions for different parties involved in the framework.

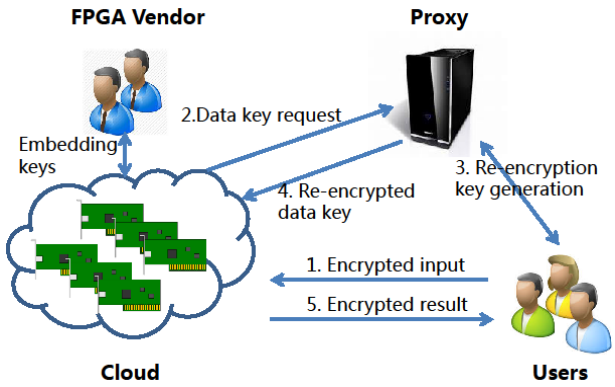


Figure 2. Overall framework of end-to-end protection of cloud based big data processing. FPGA vendor collaborates with the cloud to initialize the FPGAs, which are managed by the cloud. The users send encrypted data to cloud and the cloud processes the data using FPGAs. In the processing procedure, the proxy is set up for cipher-texts conversion. At the final step, the cloud returns the encrypted results to the users.

3.1 System Overview.

As shown in *Figure 2*, four parties are involved in FPGA cloud: *cloud user*, *FPGA vendor*, *cloud service provider*, and *proxy*.

1) Cloud user: A cloud user possesses a public/private key pair. When the user outsources the computation work to the cloud, the user first encrypts his/her data with a symmetric encryption scheme such as AES using a data key. The data key is further encrypted with user's public key. The user coordinates with the proxy for computing re-encrypted data keys;
2) FPGA vendor: FPGA vendor manufactures FPGA based computing devices that are ready to be deployed in the cloud. The vendor is also responsible for embedding keys into the FPGAs. The keys can be embedded in the FPGA bitstream that is encrypted and protected with the existing FPGA IP protection mechanisms. As mentioned in Section 2, each FPGA has two types of keys, one is a public/private key pair that is embedded into the bitstream, the other is a symmetric key, which is used to encrypt the bitstream. The symmetric key is usually stored in tamper-proof storage of the FPGA so that no one outside can access this key. Besides releasing the hardware to the cloud

service provider, a vendor does not need to interact with any other parties to fulfil its designated role;

3) Cloud service provider: A cloud service provider possesses and manages a pool of FPGA devices besides usual computing hardware that one can find in cloud such as CPUs, memories, and disks. For privacy sensitive tasks that are designed to run on the FPGAs, the cloud provider provisions user's data (protected with a secret data key inaccessible by the service provider) to the FPGA device memory and instructs the FPGA to process the user data. After finishing the computation, the FPGA returns encrypted results to the cloud;
4) Proxy: Proxy is responsible for cipher-texts conversions and re-encryption key management. It will interact with both the cloud service provider and the cloud user.

3.2 Threat Model.

We assume that cloud user is fully trusted and the user will be diligent in having all the key materials properly protected for his/her own interests.

The cloud service provider is semi-trusted, i.e., it will work fairly by following pre-defined protocols and polices but may try to learn the user's data from what it can access. The cloud service provider cannot be fully trusted due to reasons from two perspective: (i) Risks from malicious users. It is common that multiple users reside in the same cloud infrastructure, or even the same physical machine due to virtualization. Malicious users may cross the boundary to access others' data (Ristenpart et al., 2009); (ii) Risks from malicious administrators. Administrators usually have higher privileges and malicious administrators may abuse these privileges to learn users' data (Kandias, Virvilis, & Gritzalis, 2013; Toxen, 2014).

As proxy only supports limited functions and has simple storage structure, it is easy to be well protected. So we assume the proxy is trusted.

We further assume that the vendor is trusted. The bitstream is assumed to be free from backdoors or Trojans. The vendor will not collude with the cloud service provider or the proxy to compromise user's data privacy. It is worth pointing out that without colluding with the cloud service provider, the vendor itself generally has no access to the user's encrypted data or encrypted data key. Thus, the vendor cannot compromise the privacy of user's data by itself.

3.3 Goals of FPGA Cloud.

The goals of FPGA cloud can be summarized as follows:

1) Data security. User's data sending to the cloud for processing are well protected, i.e., no one except the user has access to both the original data and the final processed results;
2) Reasonable cost and generality. FPGA cloud should be able to support different applications like big data processing with real scale.

4. Privacy Preserving FPGA Cloud

In this section we provide detailed design of FPGA cloud.

4.1 Proxy Re-encryption Scheme used for FPGA Cloud.

For cloud computing one key feature is dynamic job scheduling, and proxy re-encryption plays an important role for this capability. In this paper, we use the proxy re-encryption scheme proposed in (Ateniese et al., 2006). This scheme satisfies our requirements such as unidirectional, non-interactive, conclusion safe, and non-transitive. We give a short review of this scheme. Suppose G_1, G_2 are two groups of prime order q with a bilinear pairing map $e: G_1 \times G_1 \rightarrow G_2$. $g \in G_1$ is a random generator and let $Z = e(g, g)$. The scheme works as follows:

- Key Generation. User A's key pair is of the form $pk_A = g^a, sk_A = a$, where $a \in Z_q^*$;
- Re-encryption Key Generation. If user A wants to delegate the decryption right to user B, he computes the re-encryption key $rk_{A\rightarrow B} = g^{b/a}$;
- Encryption. To encrypt a message $m \in G_2$ with publickey pk_A, a user first chooses a random number $k \in Z_q^*$, and computes the cipher-text as $c_A = (g^{ak}, mZ^k)$;
- Re-encryption. For cipher-text $c_A = (g^{ak}, mZ^k)$, the proxy with re-encryption key $rk_{A\rightarrow B} = g^{b/a}$ computes $e(g^{ak}, g^{b/a}) = Z^{bk}$ and publishes the new cipher-text $C_B = (Z^{bk}, mZ^k)$;
- Decryption. Given cipher-text $c_A = (g^{ak}, mZ^k)$, user A who possesses private key $sk_A = a$ can compute $e(g^{ak}, g^{1/a}) = Z^k$ and then recover the plain-text $mZ^k/Z^k = m$. For FPGA cloud, the Decryption algorithm is not used;
- Re-encryption Decryption. Given re-encrypted cipher-text $c_B = (Z^{bk}, mZ^k)$, user B who possesses private key $sk_B = b$ first computes $(Z^{bk})^{1/b} = Z^k$ and then recovers the plain-text m.

4.2 Bitstream Extension.

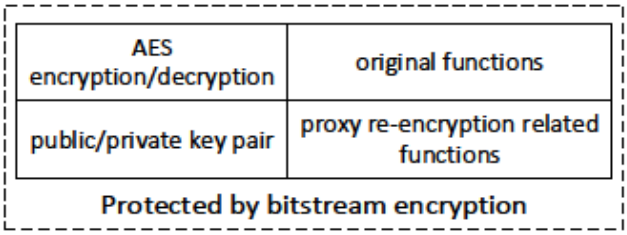


Figure 3. The extended bitstream structure. To support privacy preserving computation, extra components are added into the bitstream, e.g., the AES encryption/decryption module for input decryption and output encryption, public/private key and corresponding proxy re-encryption module for data key processing.

In order to utilize FPGA for privacy preserving computation in public cloud environment, we need to add some new features into the bitstream. *Figure 3* depicts the extended bitstream structure. Specifically, a public/private key pair related to proxy re-encryption scheme and corresponding functions are embedded into the bitstream.

4.3 Outsourcing Computation to FPGA Cloud.

In this section, we provide detailed description of using FPGA cloud to protect privacy of user's data while offering processing capacity. We assign an identity to each function type so that cloud can schedule tasks to the proper FPGAs. The FPGA cloud consists of five protocols.

System Initialization. As mentioned in Section 4.2, a pair of keys is embedded into the bitstream. For security consideration, we require that all the key pairs be unique. In other words, each FPGA i has its own key pair, denoted as (pk_i^F, sk_i^F). Each user j is also given a key pair, denoted as (pk_j^U, sk_j^U). All the public keys are published. The proxy maintains a database of proxy keys for the FPGAs and users.

TABLE 1. Database structure for storing the re-encryption keys. For each FPGA and each user, there is a re-encryption key related.

	$FPGA_1$	$FPGA_2$	…	$FPGA_n$
$user_1$	rk_{11}	rk_{12}		rk_{1n}
…	…	…	…	…
$user_m$	rk_{m1}	rk_{m2}	…	rk_{mn}

Table 1 gives an example of the proxy key database. This database stores re-encryption keys that are used for converting user encrypted data to proper form for the designated FPGA to process. The database is managed by the proxy in a lazy manner, i.e., an entry of the database is computed and saved only when the value saved in this entry is needed to perform data processing. The disadvantage of this strategy is that user has to remain online in order to cooperate with the proxy after submitting his/her task.

Data Uploading. Before uploading any data to the cloud, user j randomly generates a symmetric key *dek* to encrypt the data that needs to be processed in the FPGA cloud. *dek* is further encrypted with user *j's* public key pk_j^U. Denote the encrypted data as c_{data} and the cipher-text of *dek* as c_{dek}. User j sends c_{data}, c_{dek}, his/her identity j, and the function type identity. User also records the relationship between the symmetric key and the task.

Task Scheduling. One advantage of cloud computation is that the cloud service provider possesses a resource pool and can allocate resources to the users on demand. After receiving the user's request, the cloud has to schedule certain FPGA(s) for the task. It first checks the function type to decide which group of FPGAs can be used for this task, then schedule the job to these FPGAs. The service provider may consider different factors when choosing the FPGA, e.g., the physical location, load balancing, etc. For each chosen FPGA, the cloud service provider asks the proxy for re-encryption of the user's data for this FPGA. The proxy will query the database for the FPGAs and users. If the re-encryption key exists, then the proxy re-encrypts c_{dek} to c'_{dek} and sends c'_{dek} to the cloud provider. Then, the FPGA decrypts it with the private key embedded into its bitstream. If the re-encryption key does not exist, then the proxy interacts with the user to generate the re-encryption key, saves the key in the database, and performs the above operations. If multiple FPGAs are selected, then the cloud performs the same work for each of them. After receiving response from the proxy, the cloud service provider sends c_{data} and c'_{dek} to the FPGA for data processing.

Data Processing. With the decryption function and key embedded into the bitstream, an FPGA can decrypt c'_{dek} to get dek. Then it can further decrypt c_{data} and does the required processing procedure. Note that all the decryptions are carried out inside the FPGAs and it is not possible for the cloud service provider to find out and disclose user's data. Thus, user's data privacy cannot be compromised. After finishing the data processing, the FPGAs encrypt the results with dek and then the cloud will return them to the user.

Results Retrieval. Results retrieval is very simple. As the user knows the symmetric key dek, he/she can decrypt the results to get the plain-text of the result.

5. Security Evaluation and Enhancement of FPGA Cloud

In this section we evaluate the security property of FPGA cloud and discuss potential risks.

5.1 Security of FPGA Cloud

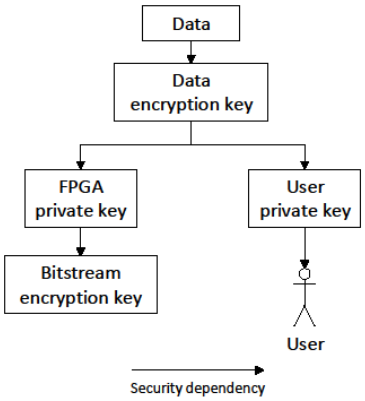


Figure 4. Security of user's data. The user uses his/her public key to encrypt dek before sending to the cloud. The proxy only has access to the re-encryption key to help on converting c_{dek} to c'_{dek}, but cannot learn any information about dek. The only place where dek exists in plain-text form is inside the FPGA. The tamper resistant property of FPGA assures the security of dek.

According to the protocol given in Section 4, except inside the tamper resistant FPGA, user's original inputs and final results are always encrypted using a symmetric key *dek* and *dek* is protected by either the user's public key or the FPGA's public key. The proxy will converting the cipher-text of *dek*, and the security property of the proxy re-encryption scheme assures the proxy cannot learn anything about *dek*. So security of the user's data only depends on the security of the private keys of the user/FPGA.

The user will not leak his/her private key to any parties. The re-encryption key generation involves user's private key but this procedure is finished on the user side. The FPGA private key is protected by the bitstream encryption scheme and depends on the security of the bitstream encryption key. *Figure 4* summarizes the security relationship between these components.

5.2 Potential Risk of FPGA Cloud and Enhancement

Initial key generation. If the symmetric keys used for bitstream protection leak to adversaries, the whole solution fails. Our solution makes an assumption that the FPGA vendor is responsible for embedding keys into FPGAs, and the FPGA vendor becomes a potential vulnerability. To overcome this challenge, Xu and Shi (2014) proposed a method to enable the FPGA vendor and the cloud service provider work together to generate the keys, so the bitstream protection keys are at risk only when both parties are compromised.

Side channel attacks. One key feature of the proposed solution is to utilize FPGAs as black boxes in the cloud. Every security solution utilizing hardware black boxes faces the threat of side channel attacks. By collecting side channel information like energy consumption (Messerges, Dabbish & Sloan, 1999, 2002), radiation (Gandolfi, Mourtel & Olivier, 2001), time (Renauld, Standaert & Veyrat-Charvillon, 2009), or even sound information (Genkin, Shamir & Tromer, 2013), the attacker can recover the secret information stored inside the hardware. Specified side channel attacks target at FPGAs are also studied (e.g., Moradi, Barenghi, Kasper & Paar, 2011; Moradi, Oswald, Paar & Swierczynski, 2013), but these attacks generally have special and rigorous requirements for the attacking environment. For example, Moradi et. Al (2011) proposed power analysis attack against Xilinx FPGA, which requires connecting the FPGA board to a special communication module and use digital oscilloscope to collect the side channel information.

In the cloud environment, users have no physical access to the data center and devices deployed there. So it is hard for malicious users to launch side channel attacks. Even though administrators usually have more control over the cloud computing system than general users, they also have very limited physical access to the infrastructure (Steiner, 2012). Thus, it is improbable for administrators to attach special devices to collect side channel information. The effectiveness of side channel attacks is heavily dependent on the hardware/software implementation of a system. Careful design and implementation of the FPGA board can make side channel attacks harder (Bogdanov, Moradi & Yalcin, 2012; Canetti & Hohenberger, 2007; Güneysu & Moradi, 2011; Poschmann et al., 2011).

Collusion between different parties. If one or more parties do not act according to its protocol, the privacy of user's data may be at risk. One potential risk is collusion between the cloud service provider and the proxy. Suppose that a user uploads data to the cloud where the data is protected with a data encryption key *dek*. Instead of using the public key of an FPGA, the cloud service provider generates a public/private key pair (pk_{fake}, sk_{fake}) with the same public parameters of the deployed proxy re-encryption scheme. Then the proxy interacts with the user and asks for a re-encryption key $rk_{u \to fake}$. If the user responds to this request and generates the re-encryption key for the proxy, the proxy can use this key to convert the cipher-text of *dek* to another cipher-text that can be decrypted using sk_{fake}. Then the cloud service provider can learn the plain-text of *dek* and use this key to decrypt user's data. This risk can be mitigated by introducing PKI, i.e., each public key is associated with a certificate that is signed by a CA. When a user receives a request of re-encryption key generation, he/she can verify whether this public key is valid or not, and only respond to these valid public keys.

6. FPGA Cloud for k-NN Clustering with MapReduce

In this section, we describe using the FPGA cloud for big data processing with MapReduce framework.

6.1 Short Review of MapReduce Framework

MapReduce is a powerful tool for big data processing (Dean & Ghemawat, 2008), and is popular for cloud based data analytics. As the name implies, MapReduce contains two main functions: *map* function and *reduce* function. The MapReduce process can be broken down into a pipeline as shown in *Figure 5* (Zhang, Cherkasova & Loo, 2013).

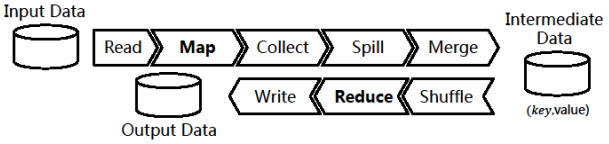


Figure 5. Original MapReduce processing pipeline. The results of mappers go through three steps and stored as intermediate results. Then the intermediate results are shuffled to different reducers.

Map task consists of five phases:

1) *Read*. Reading a data block (e.g., 64 MB) from distributed file system (e.g., Hadoop distributed file system);

2) *Map*. Applying the map function to each record in the input file and generating the map-output data (intermediate data);
3) *Collect*. Buffering the map phase outputs into memory;
4) *Spill*. Sorting and partitioning the intermediate data across different reduce tasks;
5) *Merge*. Merging different spill files into a single spill file for each reduce task.

Reduce task has the following three phases:

1) *Shuffle*. Transferring the intermediate data from map tasks to reduce tasks and merge-sorting them together;
2) *Reduce*. Applying the reduce function on the input key and all the values corresponding to it to produce the final output data;
3) *Write*. Writing the reduce output to the distributed file system.

6.2 MapReduce Task in FPGA Cloud.

A MapReduce task can be divided into three parts in FPGA cloud framework, as *Figure 6* depicted.

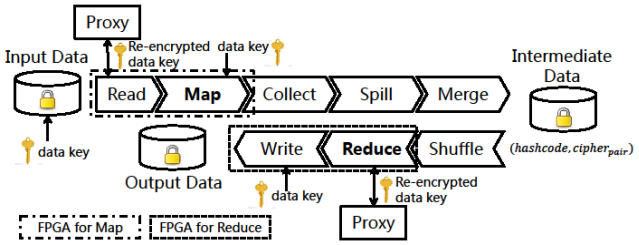


Figure 6. Secure MapReduce processing pipeline with FPGA cloud. The map function and reduce function are carried out by FPGAs. As different FPGAs have different private keys, proxy re-encryption scheme is deployed to support flexible job scheduling. Other operations (e.g., collect, spill, etc.) are not affected. The structure of intermediate results (hashcode, $cipher_{pair}$)is discussed in Section 6.3.

- FPGAs for the map function are responsible for *read* and *map*;
- FPGAs for the reduce function are responsible for *reduce* and *write*;
- General purpose processors (CPUs) are responsible for computations related to other scheduling jobs, e.g., *collect*, *spill*, *merge*, and *shuffle*.

Compared with the standard MapReduce process without privacy preserving support, the scheduling jobs carried out by the cloud service provider using CPUs are the same, and the differences are all related to the FPGA jobs. *Figure 7* summarizes the whole processing procedure. In the following description, *key* may refer to either part of the (*key*, *value*) pair in MapReduce framework or an encryption key.

1) *Input data*. The input data is encrypted by a user using a randomly generated data encryption key dek, then dek is encrypted using the user's public key, and the cipher-text c_{dek} is stored together with the encrypted data. Here, the data encryption algorithm should be compatible with the data division strategy, i.e., when the cipher-text of the data is divided into blocks, each block can be decrypted properly independently. For example, when AES is used for encryption, the data can be encrypted block by block, other than encrypted as a whole;
2) *Read*. When an FPGA for running the map function reads a block, it also reads the cipher-text of the related data encryption key. Following the protocol mentioned in Section 4, the FPGA interacts with the proxy to get a re-encrypted cipher-text of the data encryption key. Then it can decrypt to get *dek* with its own private key. After that, the data block can be decrypted;
3) *Map*. After the decrypted data block is processed, the FPGA for the map function gets a set of pairs *(key, value)*, which are the same as the standard MapReduce process. However, before sending these pairs as output, the FPGA for the map function encrypts the original pair *(key, value)* with *dek*, denoted as $cipher_{pair}$, and applies a transformation to *key* to get another index *hashcode*. The cloud service provider uses *hashcodes* for scheduling as replacement of the original reduce key. We will discuss details of this transformation in Section 6.3;

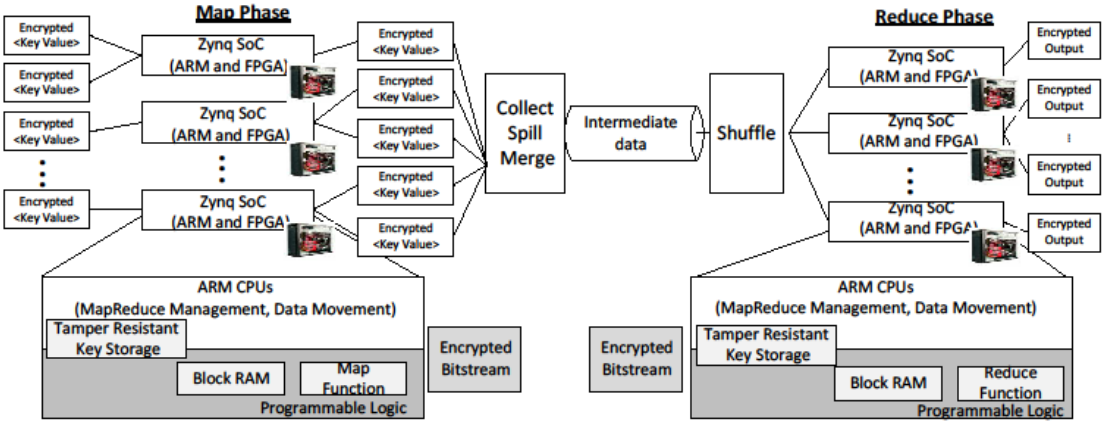


Figure 7. MapReduce with FPGA cloud. Each MapReduce node contains a Zynq board: dual core ARM + FPGA. The ARM CPU supports regular Linux and runs the software stack of MapReduce except the Map function and Reduce function. Map and Reduce functions are distributed as encrypted and certified bitstreams. During boot, they are instantiated as programmable logic. All (key, value) inputs, intermediate (key, value) outputs, and the final results are encrypted. The cryptographic key for data encryption and decryption is only visible to the FPGA. The ARM Linux has no access to the plain-texts of the (key, value) pairs.

htt

4) *Reduce*. After receiving a pair $(hashcode, cipher_{pair})$, an FPGA for running the reduce function interacts with the proxy and asks for a re-encrypted cipher-text of dek. Then the reducer FPGA decrypts to get dek, and uses dek to decrypt $cipher_{pair}$. After that, the reduce function is carried out and the final result is calculated;
5) *Write*. Before writing the result to the distributed file system, the reducer FPGA encrypts the result with dek. User who uploads the task knows dek so the user can decrypt and extract the final result.

6.3 Statistic Information Leakage and Counter Measures.

For MapReduce framework with FPGA cloud, the mapper FPGA cannot encrypt the whole *(key, value)* pairs directly because the cloud service provider needs this information of *keys* to schedule these pairs to reducer FPGAs. The basic principle of MapReduce is that output pairs of mappers with the same *key* should be dispatched to the same reducer FPGA. Encryption of the *value* part of each pair is not enough, as *key* usually reflects some property of *value*, and attacker may gain useful information even if all *values* are encrypted. Take the application of word count as an example, the output of *mapper* is in the form *(key, 1)*, where *key* is the word appeared in the data being processed. It is easy to see that encrypting the *value* part does not help us to protect the data being processed. Hash function can be applied to *key* to hide the original word but it does not help for the protection of the distribution. Attackers who can access the intermediate result may easily obtain the word frequency information of the input data. Another potential risk is that if the domain of *value* is small and *key* is generated from *value* without adding any secret, one may use the dictionary attack to recover plain-text of *value* with the *key*.

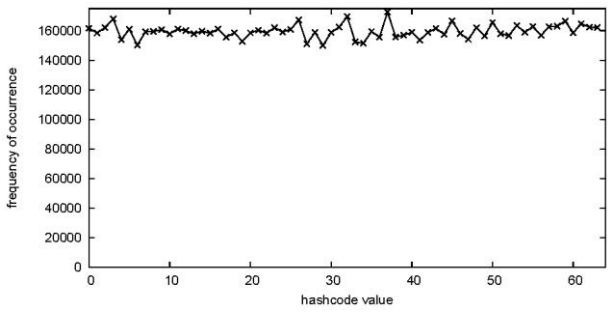


Figure 8. 6 bits hashcode distribution for a dataset of URLs given in (Leskoveca et al., 2009).

In order to prevent such risks, we propose using the truncated HMAC to generate a *hashcode* from the original *key*, and *dek* is used as secret key to HMAC. As a secret key is involved in the process of calculating *hashcode* , dictionary attack is prevented. To verify the effectiveness of our method against frequency attack, we apply truncated HMAC to a data set that contains more than 5 million url items (Leskoveca, Langb, Dasguptab & Mahoneya, 2009) in simulated MapReduce PageRank test and keep the first 6 bits of the HMAC values as *hashcode*. *Figure 8* shows the frequency of different *hashcode* s. It can be seen that although several *hashcode*s appear more than others, the overall distribution is even, which means an attacker cannot learn much from observing the frequency information. Furthermore, because we use HMAC to generate *hashcodes*, there are little restrictions of the input format, which makes our method universal.

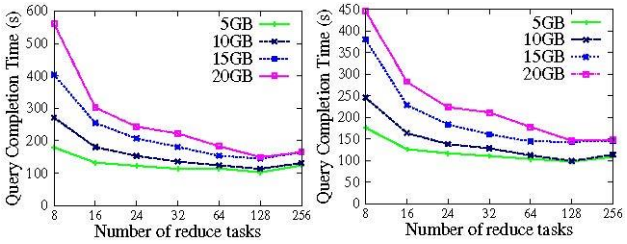
For certain data sets and applications, after applying our *hashcode* generation method, the distribution of the result *hashcode*s could be uneven and still leak some information. In this case, the mapper FPGA can choose to add pseudo key value pairs into the real pairs to make the distribution even and mitigate the possibility of statistical attack.

Another issue is the length of the $cipher_{pair}$. If the original pairs *(key, value)* have different sizes, the corresponding cipher-text also have different lengths, which may cause potential information leakage. A straightforward solution to solve this problem is to add some constant data to the original pair before encryption so that the cipher-texts always have the same length.

6.4 Security Analysis and Evaluation of MapReduce with FPGA Cloud.

Combining the *hashcode* protection method together with FPGA cloud, user's data is properly protected: the plaintexts of user's data only appear inside the FPGAs for map and reduce, and the cloud service provider has no access to the data. Furthermore, application of *hashcode* makes it is hard for an adversary to learn statistics information about user's data.

By applying the *hashcode* protection method, the



(a) TPC-H Q1 *(b)TPC-H Q19*

Figure 9. Execution time under different number of reduce tasks (Zhang et al., 2013). It can be seen that increasing the number of reducers cannot always reduce the computation time. Thus, the length of hashcode can be small, which assures the evenly distribution property. TPC-H is a benchmark containing different types of queries. Q1 is pricing summery report query and Q19 is discounted revenue query.

freedom degree of scheduling is affected. Specifically, the bit number of *hashcode* sets an upper bound of potential FPGAs as reducers. For example, if the bit number is 6, then no more than $2^6 = 64$ FPGAs can be used as reducers. In practice, this restriction will not affect the overall MapReduce performance very much (Zhang et al., 2013), because MapReduce execution time does not decrease linearly with the increasing number of FPGA reducers. *Figure 9* shows the relationship between execution time and number of reducer on different size datasets. From this figure, we can see that 64 reduce tasks (6 bits of *hashcode*) is a good choice in practice for reasonable size of data. When more reducers are needed, one can raise the upper bound by increasing the *hashcode* length.

7. K-NN CLUSTERING WITH FPGA CLOUD AND MAPREDUCE

In this section we describe applying FPGA cloud for k-NN, a concrete application of MapReduce. Implementation details and experimental evaluation are also provided.

7.1 SHORT REVIEW OF K-NN CLUSTERING

The k-nearest neighbors algorithm (k-NN) is a nonparametric method used for classification. Specifically, k- NN finds a group of k objects in the training set that are closest to the test object, and bases the assignment of a label on the predominance of a particular class in this neighborhood (Wu et al., 2008). Algorithm 1 describes the basic version of k-NN clustering algorithm.

Algorithm 1 k-NN clustering

Input: The training objects set $T=\{(x, y)\}$ (x is the data point and y is the class label), the test data point x', and the parameter k ($|D| \geq k$);

Output: the class label of x';

1: $D \leftarrow \emptyset$;
2: **for** each $(x, y) \in T$, where x is the data point and y is the class label **do**
3: $(d, y) \leftarrow dis(x, x')$;
4: $D \leftarrow D \cup (d, y)$;
5: **end for**
6: Find $D' \subset D$ such that $|D'| = k$, and every $(d', y') \in D'$ satisfies $d' < d$ where $(d, y) \in D - D'$;
7: Find class label y' that appears most in D';
8: **return** y';

k-NN clustering algorithm can be ported to MapReduce framework:

- Map function. For a test data point, the map function calculates the distance between the test data point and all the data points in the training set. The test data is used as *key*, *value* contains the distance information and the class label;
- Reduce function. For a given *key*, the reduce function first selects k key-value pairs that have the smallest distance. Then the reduce function checks the k class labels.

7.2 IMPLEMENTATION OF K-NN WITH FPGA CLOUD

The implementation of k-NN clustering uses ZedBoard, which is based on Xilinx Zynq-7000 SoC (xc7z020clg484-1). ZedBoard mainly consists of two modules, the FPGA component and an ARM core, which can run Linux. To convert *C*-based functions of mapper and reducer to hardware modules, Xilinx Vivado High Level Synthesis (HLS) is utilized. We set up PIPELINE directive in Vivado HLS to archive a higher throughput and a lower latency by adding pipeline stages in hardware.

Key related implementation. Proxy re-encryption scheme in (Ateniese et al., 2006) is used to support dynamic job scheduling, and bilinear pairing is the most expensive part of this scheme. Bilinear pairing is usually constructed with elliptic curve points group and calculated using Miller's (2004) algorithm. The parameters we use here are as follows: the elliptic curve is $E: y^2 = x^3 - x + 1$, which is super-singular. The underlining finite field is $GF(3^{97})$, where the irreducible polynomial is $x^{97} + x^{12} + 2$. Here G_1 is a subgroup of the points on E, and G_2 is a cyclic subgroup of $GF(3^{582})$. The reason we choose these parameters is that in terms of bandwidth efficiency, it is more efficient to use elliptic curves in characteristic three for systems based on bilinear pairing such as Weil or Tate pairing (Galbraith, 2001), and lots of work has been done to optimize bilinear pairing computation with finite field with character three (Beuchat et al., 2008; Kerins, Marnane, Popovici & Barreto, 2005; Iwan & Hyang-Sook, 2003).

On the user and proxy side, *encryption* and *re-encryption* functions are implemented based on Miracl library (http://www.certivox.com/miracl). The FPGA only needs to support *re-encryption decryption*, which equals to one division in $GF(3^{582})$. We adopt the implementation of (Page & Smart, 2003) for this function.

Key-value pair protection/processing implementation. The key-value pair protection consists two components: AES encryption/decryption and HMAC evaluation. An open IP core from the OpenCore.org is used for the AES encryption/decryption. This implementation takes around 30 cycles to encrypt/decrypt a 128-bit data block (operating at 100MHz). HMAC is used for *hashcode* generation. Like AES, FPGA implementations of HMAC have been studied a lot in the past and our implementation is based on the HMAC module of (Juliato & Gebotys, 2011). The resources consumption of these two modules is summarized in *Table 2*.

For data processing, the number of clusters does not affect to resource consumption of map function, while dimensions of data point does not cause any change in reducer's hardware resource, as shown in *Table 3*.

Table 2. Resources allocated for AES with key length of 128 bits (both encryption and decryption) and HMAC using SHA2-256.

	DSP48E	FF	LUT	BRAM
AES	0	6482	4548	50
HMAC	0	1818	3950	2

TABLE 3. Resources allocated for map and reduce function with different parameters. k is the number of clusters, and d is the dimension of data point.

	DSP4E	FF	LUT
map $(d = 4)$	4	7	111
map $(d = 8)$	4	79	115
map $(d = 16)$	4	80	119
reduce $(d = 32)$	0	118	213
reduce $(d = 64)$	0	121	219
reduce $(d = 128)$	0	124	225

In our implementation, a pair of AES encryption/decryption with map/reduce cores and an HMAC core are coupled (*Figure 10* shows the map function).We denote the map function coupled with AES and HMAC *map slice*, and reduce function with AES and HMAC *reduce slice*. Because the map core runs 2 to 3 times faster than the AES core, we allocate multiple AES cores per map core in order to get higher throughput for the system as a whole. Resource consumptions for the *map/reduce slices* are presented in the *Table 4*. The column "Slices per ZedBoard" shows the number of *map/reduce slices* we can allocate on a ZedBoard based on resource availability.

TABLE 4. Resources allocated for map and reduce slices with AES and HMAC cores.

	DSP48E	FF	LUT	BRAM	Slices / board
map $(d = 4)$	4	27824	22253	204	1
map $(d = 8)$	4	27825	22257	204	1
map $(d = 16)$	4	27826	22261	204	1
reduce $(k = 32)$	0	14900	13259	104	2
reduce $(k = 64)$	0	14903	13265	104	2
reduce $(k = 128)$	0	14906	13271	104	2

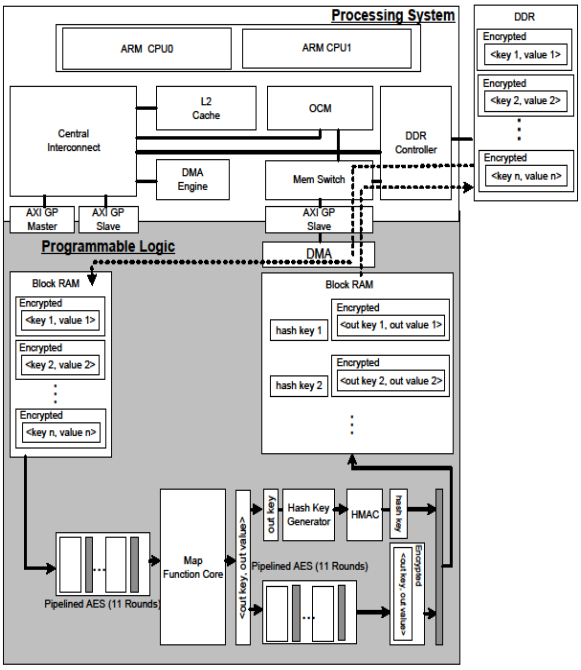


Figure 10. Secure Map function core for k-NN executed on Zynq FPGA device: the ARM Linux is responsible for everything except running the Map function kernel. Encrypted test points and training objects are stored in the input BRAM using DMA transactions. The Map function core decrypts the data and calculates the distances. Output key-value pairs are encrypted and stored in the output BRAM. A cryptographic hash key is computed using the output key as input. The ARM CPU will copy the encrypted output data from BRAM to DDR using DMA transactions. Temporary results of the Map function logic are stored in LUT-RAM which is invisible to the ARM. There are multiple I

DDR mem

existing

transactio

TABLE 5. Storage cost for re-encryption key management.[1]

#FPGA / #user	1k	5k	10k
100	4 MB	20 MB	40 MB
500	20 MB	100 MB	200 MB
1000	40 MB	200 MB	400 MB

[1] *For simplicity, we use 194 bits for an element of GF*(3^{97}).

7.3 Experimental Results of k-NN with FPGA Cloud

Key related performance. The proxy stores all the re-encryption keys for job scheduling. *Table 5* shows the relationship between the storage cost and the number of users/FPGAs in the system. The storage cost given in *Table 5* is the upper bound as it assumes there is a re-encryption key for each user and FPGA. For moderate parameters, the re-encryption keys can be easily put into the proxy's memory.

For a k-NN task, the user may use a single data key *dek* for all the data so each FPGA only needs to decrypt c'_{dek} once, and time cost is negligible. The main time consuming parts are cipher-text re-encryption (on the proxy side) and re-encryption key generation (on the user side). *Figure 11* depicts the cost of these two parts. The burden on user side will decrease as more re-encryption keys are established.

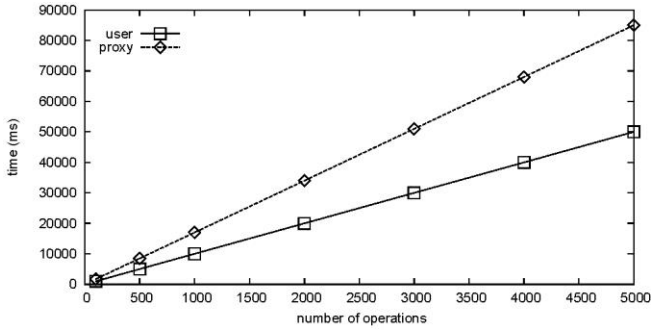
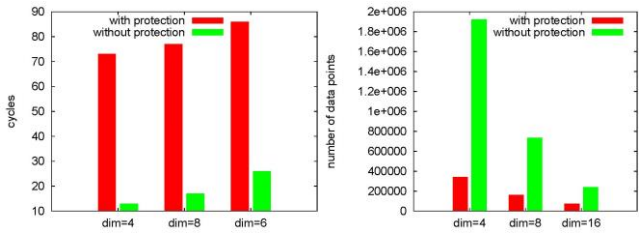
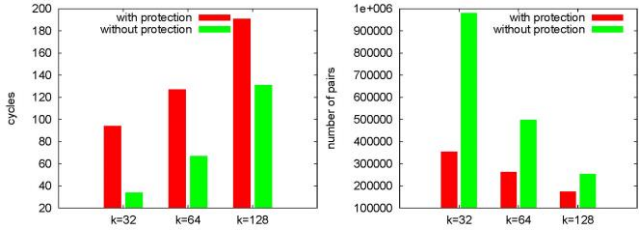


Figure 11. Time cost related to proxy re-encryption. The computation burden on user side is re-encryption key generation and on the proxy side it is the cipher-text re-encryption.

Key-value pair protection/processing performance. The performance of k-NN with FPGA cloud is affected by two parameters: the data point dimension *d* and the number of neighbors *k*. *Figure 12* provides latency/throughput information of map and reduce module. As the map/reduce functions of k-NN are very simple, AES module becomes the performance bottle neck. As the ZedBoard we use for experiments only has limited resources, we cannot deploy more AES modules. Otherwise more AES modules can be coupled with the map/reduce module to achieve lower latency and higher throughput.



(a) Latency of map function. *(b) Throughput of map function.*



(a) Latency of reduce function. *(b) Throughput of reduce function.*

Figure 12. Performance of the map and reduce functions in FPGA cloud.

8. Related Work

One research area of cloud computing that has received a lot of attention recently is to utilize cryptographic tools to provide secure outsourced computation. Some research efforts focus on specialized applications such as outsourced databases (Hacigümüş et al., 2002). Others attempt to enable pattern matching and/or text searching on cipher-text (e.g., (Boneh et al., 2004; Bellare et al., 2007; Song et al., 2000)). By now, the most powerful tool in terms of cryptographic protection is fully homomorphic encryption (Gentry, 2009) that supports arbitrary operations on cipher-text. Many of these approaches have significant theoretical contributions. However, it is infeasible to apply them in the real world because they either support limited application scenarios or suffer from high computation/storage cost. There is a great need to create practical privacy preserving solutions for big data analytical applications such as those based on the parallel MapReduce programming model. Such a need has not been met by the existing solutions in the literature.

From the perspective of secure FPGA, a lot of research has been done to protect the FPGA bitstream itself. Some researchers have proposed using physical unclonable functions (PUFs) to protect the FPGA bitstream (Guajardo, Kumar, Schrijen & Tuyls, 2007; Kumar, Guajardo, Maesyz, Schrijen & Tuyls, 2008). Other techniques such as watermarking and signature are also used to protect the FPGA IPs (Kahng, Lach & Mangione-Smith, 2001; Lach, Mangione-Smith & Potkonjak, 1999; Schmid, Ziener & Teich, 2008). The main objective of these prior research efforts is to protect the intellectual property of the FPGA bitstream developers, which is to make sure that the bitstream cannot be either reverse-engineered or illegally duplicated. These IP protection techniques cannot be applied directly to solve the problem of data privacy in the cloud.

There are recent efforts on applying FPGAs as accelerators for parallel data analytics (Court et al., 2004; Ronan et al., 2006; Woods & VanCourt, 2008). Shan et.al present a framework to use FPGAs to accelerate MapReduce processing in (Shan et al., 2010). All these related efforts do not consider the problem of data privacy.

9. Conclusion

This paper proposes an FPGA cloud framework that supports privacy preserving computation outsourcing. The FPGA cloud uniquely integrates proxy re-encryption with FPGAs so that users can utilize the capabilities of cloud computing while keeping the privacy of their data. Compared with existing solutions, the FPGA cloud achieves its security goal at reasonable cost. Besides straight forward applications such as signal processing, the paper discusses applying FPGA cloud to support MapReduce. Furthermore, we conduct experiments and evaluation to demonstrate the practicability and effectiveness of the FPGA cloud with k-NN clustering.

10. ACKNOWLEDGEMENT

The authors would like to thank all the reviewers for their valuable comments and suggestions to improve the quality of the paper.

11. REFERENCES

Ateniese, G., Fu, K., Green, M., and Hohenberger, S. (2006). Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Transactions on Information and System Security, 9:1–30.

Bellare, M., Boldyreva, A., and O'Neill, A. (2007). Deterministic and e_ciently searchable encryption. In Menezes, A., editor, Advances in Cryptology – CRYPTO 2007, volume 4622 of LNCS. Springer.

Beuchat, J.-L., Brisebarre, N., Detrey, J., Okamoto, E., Shirase, M., and Takagi, T. (2008). Algorithms and arithmetic operators for computing the _T pairing in characteristic three. IEEE Transactions on Computers, 57(11):1454–1468.

Blaze, M., Bleumer, G., and Strauss, M. (1998). Divertible protocols and atomic proxy cryptography. In Goos, G., Hartmanis, J., and van Leeuwen, J., editors, Advances in Cryptology - EUROCRYPT 1998, volume 1403 of LNCS, pages 127–144. Springer.

Bogdanov, A., Moradi, A., and Yalcin, T. (2012). Effcient and side-channel resistant authenticated encryption of fpga bitstreams. In ReConFig, pages 1–6.

Boldyreva, A., Chenette, N., Lee, Y., and O'Neill, A. (2009). Order-preserving symmetric encryption. In Joux, A., editor, Advances in Cryptology - EUROCRYPT 2009, volume 5479 of LNCS, pages 224–241. Springer.

Boldyreva, A., Chenette, N., and O'Neill, A. (2011). Order-preserving encryption revisited: Improved security analysis and alternative solutions. In Rogaway, P., editor, Advances in Cryptology - CRYPTO 2011, volume 6841 of LNCS, pages 578–595. Springer.

Boneh, D., Crescenzo, G. D., Ostrovsky, R., and Persiano, G. (2004). Public key encryption with keyword search. In Advances in Cryptology - EUROCRYPT 2004, volume 3027 of LNCS, pages 506–522. Springer.

Brakerski, Z. and Vaikuntanathan, V. (2011). Effcient fully homomorphic encryption from (standard) LWE. In Ostrovsky, R., editor, IEEE 52nd Annual Symposium on Foundations of Computer Science - FOCS 2011, pages 97-106. IEEE Computer Society.

Canetti, R. and Hohenberger, S. (2007). Chosenciphertext secure proxy re-encryption. In di Vimercati, D. C., Sabrina, and Syverson, P., editors, Proceedings of the 14th ACM conference on Computer and communications security - CCS 2007, pages 185–194. ACM.

Court, T. V., Gu, Y., and Herbordt, M. (2004). FPGA acceleration of rigid molecule interactions. In Becker, J., Platzner, M., and Vernalde, S., editors, Field Programmable Logic and Application, volume 3203 of LNCS, pages 862–867. Springer.

Curino, C., Jones, E. P. C., Popa, R. A., Malviya, N., Wu, E., Madden, S., Balakrishnan, H., and Zeldovich, N. (2011). Relational cloud: a database service for the cloud. In Fifth Biennial Conference on Innovative Data Systems Research - CIDR 2011, pages 235–240.

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. Communication of the ACM, 51(1):107–113.

Galbraith, S. D. (2001). Supersingular curves in cryptography. In Boyd, C., editor, Advances in Cryptology-ASIACRYPT 2001, volume 2248 of LNCS, pages 495-513. Springer.

Gandolfi, K., Mourtel, C., and Olivier, F. (2001). Electromagnetic analysis: Concrete results. In Çetin K. Koç, Naccache, D., and Paar, C., editors, Cryptographic Hardware and Embedded Systems - CHES 2001, volume 2162 of LNCS, pages 251–261. Springer.

Genkin, D., Shamir, A., and Tromer, E. (2013). Rsa key extraction via low-bandwidth acoustic cryptanalysis. Cryptology ePrint Archive, Report 2013/857. http: //eprint.iacr.org/.

Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In Mitzenmacher, M., editor, Proceedings of the 41st Annual ACM Symposium on Theory of Computing - STOC 2009, pages 169–178. ACM.

Guajardo, J., Kumar, S. S., Schrijen, G.-J., and Tuyls, P. (2007). FPGA intrinsic PUFs and their use for IP protection. In Paillier, P. and Verbauwhede, I., editors, Cryptographic Hardware and Embedded Systems – CHES 2007, volume 4727 of LNCS, pages 63–80. Springer.

Güneysu, T. and Moradi, A. (2011). Generic sidechannel countermeasures for reconfigurable devices. In Cryptographic Hardware and Embedded Systems–CHES 2011, pages 33–48. Springer.

Hacigümü ş, H., Iyer, B. R., Li, C., and Mehrotra, S. (2002). Executing SQL over encrypted data in the database-service-provider model. In Franklin, M. J., Moon, B., and Ailamaki, A., editors, Proceedings of the ACM International Conference on Management of Data -SIGMOD 2002, pages 216–227. ACM.

Iwan, D. and Hyang-Sook, L. (2003). Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In Chi-Sung, L., editor, Advances in Cryptology – ASI-ACRYPT 2003, volume 2894 of LNCS, pages 111–123. Springer.

Juliato, M. and Gebotys, C. (2011). FPGA implementation of an HMAC processor based on the SHA-2 Family of hash functions. Technical report, University of Waterloo.

Kahng, A. B., Lach, J., and Mangione-Smith, W. H. (2001). Constraint-based watermarking techniques for design ip protection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 20(10):1236-1252.

Kandias, M., Virvilis, N., and Gritzalis, D. (2013). The insider threat in cloud computing. In Critical Information Infrastructure Security, pages 93–103. Springer.

Kerins, T., Marnane, W. P., Popovici, E. M., and Barreto, P. S. L. M. (2005). E_cient hardware for the tate pairing calculation in characteristic three. In Rao, J. R. and Sunar, B., editors, Cryptographic Hardware and Embedded Systems - CHES 2005, volume 3659 of LNCS, pages 412–426. Springer.

Kumar, S. S., Guajardo, J., Maesyz, R., Schrijen, G.-J., and Tuyls, P. (2008). Extended abstract: The butterfly PUF protecting IP on every FPGA. In Tehranipoor, M. and Plusquellic, J., editors, IEEE International Workshop on Hardware-Oriented Security and Trust - HOST 2008., pages 67–70. IEEE.

Lach, J., Mangione-Smith, W. H., and Potkonjak, M. (1999). Robust FPGA intellectual property protection through multiple small watermarks. In Irwin, M. J., editor, Proceeding of the 36th Design Automation Conference, pages 831–836.

Leskoveca, J., Langb, K. J., Dasguptab, A., and Mahoneya, M. W. (2009). Community structure in large networks: Natural cluster sizes and the absence of large welldefined clusters. Internet Mathematics, 6:29–123.

Li, J., Zhang, D., Qiu, M., Zhu, Y., and Shen, J. (2011). Security protection on fpga against di_erential power analysis attacks. In Sheldon, F. T., Abercrombie, R. K., and Krings, A. W., editors, Proceedings of the Seventh AnnualWorkshop on Cyber Security and Information Intelligence Research, page 67. ACM.

Libert, B. and Vergnaud, D. (2008). Unidirectional chosen-ciphertext secure proxy re-encryption. In Cramer, R., editor, Public Key Cryptography -PKC 2008, volume 4939 of LNCS, pages 360–379. Springer.

Liu, L., Kantarcioglu, M., and Thuraisingham, B. (2009). Privacy preserving decision tree mining from perturbed data. In Jr., R. H. S., editor, 42nd Hawaii International Conference on System Sciences – HICSS 2009, pages 1–10. IEEE.

McNeil, S. (2012). Solving today's design security concerns. Technical report, Xinlinx.

Mell, P. and Grance, T. (2009). The NIST definition of cloud computing - NIST SP 800-145.

Messerges, T. S., Dabbish, E. A., and Sloan, R. H. (1999). Power analysis attacks of modular exponentiation in smartcards. In Çetin K. Koç and Paar, C., editors, Cryptographic Hardware and Embedded Systems – CHES 1999, volume 1717 of LNCS, pages 144–157. Springer.

Messerges, T. S., Dabbish, E. A., and Sloan, R. H. (2002). Examining smart-card security under the threat of power analysis attacks. IEEE Transactions on Computers, 51(1):541–552.

Microsemi (2013). Introduction to the SmartFusion2 and IGLOO2 Security Model.

Miller, V. S. (2004). The weil pairing, and its efficient calculation. Journal of Cryptology, 17(4):235–261.

Moradi, A., Barenghi, A., Kasper, T., and Paar, C. (2011). On the vulnerability of FPGA bitstream encryption against power analysis attacks: Extracting keys from Xilinx Virtex-II FPGAs. In Chen, Y., Danezis, G., and Shmatikov, V., editors, Proceedings of the 18th ACM Conference on Computer and Communications Security - CCS 2011, pages 111–124, New York, NY, USA. ACM.

Moradi, A., Oswald, D., Paar, C., and Swierczynski, P. (2013). Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering. In Hutchings, B. L. and Betz, V., editors, Proceedings of the ACM/SIGDA International symposium on Field Programmable Gate Arrays - FPGA 2013, pages 91–100. ACM.

Page, D. and Smart, N. (2003). Hardware implementation of finite fields of characteristic three. In Kaliski, B. S., Ã˘getin K. KoÃ˘g, and Paar, C., editors, Cryptographic Hardware and Embedded Systems – CHES 2002, volume 2523 of LNCS, pages 529–539. Springer.

Poschmann, A., Moradi, A., Khoo, K., Lim, C.-W., Wang, H., and Ling, S. (2011). Side-channel resistant crypto for less than 2,300 ge. Journal of Cryptology, 24(2):322 345.

Renauld, M., Standaert, F.-X., and Veyrat-Charvillon, N. (2009). Algebraic side-channel attacks on the AES: Why time also matters in DPA. In Clavier, C. and Gaj, K., editors, Cryptographic Hardware and Embedded Systems-CHES 2009, volume 5747 of LNCS, pages 97–111. Springer.

Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. (2009). Hey, you, get o_ of my cloud: Exploring information leakage in third-party compute clouds. In Jha, S. and Keromytis, A., editors, Proceedings of the 16th ACM conference on Computer and communications security-CCS 2009, pages 199–212. ACM.

Ronan, R., h Éigeartaigh, C. O., Murphy, C., Scott, M., and Kerins, T. (2006). FPGA acceleration of the Tate pairing in characteristic 2. In IEEE International Conference on Field Programmable Technology - FPT 2006., pages 213–220.

Schmid, M., Ziener, D., and Teich, J. (2008). Netlistlevel IP protection by watermarking for LUT-based FPGA s. In El-Ghazawi, T., Chang, Y.-W., Huang, J.-D., and Saha, P., editors, International Conference on Field-Programmable Technology - FPT 2008, pages 209–216.

Shan, Y., Wang, B., Yan, J., Wang, Y., Xu, N., and Yang, H. (2010). FPMR: MapReduce framework on FPGA. In Cheung, P. Y. K. and Wawrzynek, J., editors, Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays – FPGA 2010, pages 93–102, Monterey, California, USA. ACM.

Song, D. X., Wagner, D., and Perrig, A. (2000). Practical techniques for searches on encrypted data. In Proceedings of the IEEE Symposium on Security and Privacy-IEEE S&P 2000, pages 44–55. IEEE Computer Society.

Steiner, T. (2012). An introduction to securing a cloud environment. Technical report, SANS Institute.

Tang, Q. (2008). Type-based proxy re-encryption and its construction. In Chowdhury, D. R., Rijmen, V., and Das, A., editors, Progress in Cryptology – INDOCRYPT 2008, volume 5365 of LNCS, pages 130–144. Springer.

Thuraisingham, B., Khadilkar, V., Gupta, A., Kantarcioglu, M., and Khan, L. (2010). Secure data storage and retrieval in the cloud. In 6th International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2010, pages 1–8. IEEE.

Toxen, B. (2014). The NSA and Snowden: Securing the all-seeing eye. Communications of the ACM, 57:44–51.

van Dijk, M., Gentry, C., Halevi, S., and Vaikuntanathan, V. (2010). Fully homomorphic encryption over the integers. In Gilbert, H., editor, Advances in Cryptology-EUROCRYPT 2010, volume 6110 of LNCS, pages 24–43. Springer.

Woods, N. A. and VanCourt, T. (2008). FPGA acceleration of quasi-monte carlo in finance. In International Conference on Field Programmable Logic and Applications-FPL 2008, pages 335–340. IEEE.

Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z.-H., Steinbach, M., Hand, D. J., and Steinberg, D. (2008). Top 10 algorithms in data mining. Knowledge and Information Systems, 14(1):1–37.

Xu, L. and Shi, W. (2014). Removing the root of trust: Secure oblivious key establishment for FPGAs. In IEEE Computer Society Annual Symposium on VLSI – ISVLSI 2014, pages 160–165. IEEE.

Zhang, Z., Cherkasova, L., and Loo, B. T. (2013). Autotune: Optimizing execution concurrency and resource usage in MapReduce workflows. In Kephart, J. O., Pu, C., and Zhu, X., editors, 10th International Conference on Autonomic Computing - ICAC 2013, pages 175 – 181. USENIX.

Authors

Lei Xu received the B.Sc degree in Applied Mathematics from Hebei University, China, in 2004, and the Ph.D. of Computer Science from Institute of Software, Chinese Academy of Sciences, in 2011. He is currently a postdoctoral researcher at University of Houston. From 2011 to 2013, he worked as a research engineer at the Central Research Institute, Huawei Technologies Co. Ltd. His research interests include cloud computing and big data security, applied cryptography, and algebraic algorithms.

Khoa Dang Pham is working for the Department of Computer Science, University of Houston as a Research Assistant. With a 2-year experience on digital system and FPGA design, his expertise spreads from serial communication design, digital filter design, complex algorithm implementation to abstractions for reconfigurable fabric development. He received Bachelor

degree in the Mechanical Engineering from the Ho Chi Minh City University of Technology, Vietnam in 2008 and Master degree in the School of Computer Engineering from Nanyang Technological University, Singapore in 2014 respectively.

Hanyee Kim received the B.S. degree in computer science education from Korea University, Seoul, Korea, in 2012. He is a graduate student in the combined M.S and Ph.D. program at Korea University. His research interests include embedded systems, computer architecture, parallel computer architecture and programming, many-core, and computer science education.

Weidong Shi received his Ph.D. of Computer Science from Georgia Institute of Technology where he did research in computer architecture and computer systems. He was previously a senior research staff engineer at Motorola Research Lab, Nokia Research Center, and co-founder of a technology startup. Currently, he is employed as an assistant professor by University of Houston. In the past, he contributed to design of multiple Nvidia platform products and was credited to published Electronic Art console game. In addition, he authored and co-authored over publications covering research problems in computer architecture, computer systems, multimedia/graphics systems, mobile computing, and computer security. He has multiple issued and pending USPTO patents.

Taeweon Suh is an associate professor in the Graduate School of Information Security, Korea University. Prior to joining academia, he was a systems engineer at Intel Corporation in Hillsboro, Oregon, USA. His research interests include embedded systems, computer architecture, multiprocessor and virtualization. He has a BS in Electrical Engineering from the Korea University, Korea, and an MS in Electronics Engineering from the Seoul National University, Korea, and PhD in Computer Engineering from the Georgia Institute of Technology, USA.