# Active data-centric framework for data protection in cloud environment

Lingfeng Chen, Doan B. Hoang
Advance Research in Networking Laboratory
iNEXT-Centre for Innovation in IT Services and Applications
University of Technology, Sydney
Broadway NSW 2007 Australia
Email: Lingfeng.Chen@uts.edu.au, Doan.Hoang@uts.edu.au

## Abstract

*Cloud computing is an emerging evolutionary computing model that provides highly scalable services over high-speed Internet on a pay-as-usage model. However, cloud-based solutions still have not been widely deployed in some sensitive areas, such as banking and healthcare. The lack of widespread development is related to users' concern that their confidential data or privacy would leak out in the cloud's outsourced environment. To address this problem, we propose a novel active data-centric framework to ultimately improve the transparency and accountability of actual usage of the users' data in cloud. Our data-centric framework emphasizes "active" feature which packages the raw data with active properties that enforce data usage with active defending and protection capability. To achieve the active scheme, we devise the Triggerable Data File Structure (TDFS). Moreover, we employ the zero-knowledge proof scheme to verify the request's identification without revealing any vital information. Our experimental outcomes demonstrate the efficiency, dependability, and scalability of our framework.*

**Keywords**

Cloud Computing, Active data-centric, Zero-knowledge proof, Transformation, Tamper-proof

## INTRODUCTION

In recent years, the products and services around cloud computing have sprung up. Consumers can take advantage of this novel computing paradigm to achieve ultra-large-scalable computing and storage, real-time resource sharing, and convenient and productive services on a pay-as-usage basis over the Internet. However, this growing IT market is suffering from some security threats and concern of users who are worried about losing control of their own data, especially in some security and privacy sensitive application domains. Cloud computing adopts a functional segregation mechanism between resource consumed and user's devices to achieve outsourced computing and data management. This paradigm abstracts the provided services from the users. The users may not know who, what, where, when, how, and why their data is accessed or moved in the cloud service providers' (CSPs) environments (Ko et al. 2011a). Some researches and critics (Grove 2011; Lin and Squicciarini 2010; Wang et al. 2010; Yu et al. 2010) emphasized that the core data protection issues of cloud computing are related to controllability, security, and privacy. Conventional data protection schemes such as data dispersal storage (Wang et al. 2009; Wang et al. 2011), designated cryptographic mechanism (Goh et al. 2003; Kallahalla et al. 2003; Yu et al. 2010), third party trust management (Abawajy 2011; Firdhous et al. 2011; Wang et al. 2010) and policy-based access control (Goyal and Mikkilineni 2009; Lin and Squicciarini 2010; Takabi and Joshi 2012), etc. are able to perform integrity guarantee of data, or encrypt data with a public key infrastructure, or verify access behaviour correctness from a third party trust manager to achieve data safety and integrity. However, these solutions lack appropriate schemes to address accountability, auditability, and transparency which are considered the main factors to solve who-what-how-why related questions (Ko et al. 2011b; Sundareswaran et al. 2012). Moreover, in cloud environments, highly complex and dynamic hierarchical service chain determines that data handling may be delegated from a direct CSP to another different CSPs in a flexible manner (Sundareswaran et al. 2011), but often these CSPs do not employ the same protection schemes and standards (Foster et al. 2008).

To address the above problems, we propose an innovative approach that satisfies the requirement of highly decentralized cloud environment. Instead of paying attention to the attacks or violations themselves, we focus on our target data, and equip it with self-describing and self-defending capability. Our proposed framework combines preventive scheme (strong enforcement of access control and integrity verification) and detective scheme (logging data usage record and alerting violation). Moreover, the framework emphasizes "active" feature which packages raw data with active properties that enforce data usage with active defending and protection capability. The raw user's data is required to transform to our novel Triggerable Data File Structure (TDFS) before storage in the cloud. TDFS is independent of cloud third parties auditor and able to self-defend in untrusted or compromised cloud hosts. Simultaneously, it remains in cooperation constantly with cloud third parties auditor to achieve a federate protection requirement. Any access to our TDFS will trigger an automated

and authenticated process inside the data. This strong enforcement also triggers log information collection procedure (which records every access to the specific data items) that will be utilized to improve transparency and accountability of data usage via sending the log information to the data owner. To achieve the tamper-proof feature, we develop a *shell* program which is executed prior to accessing data *core*. The *shell* and the *core* constitute our framework's protection scheme. The *shell* takes charge of detection of tampering, verification and identification of request, remote communication, real-time logging, and terminate the program when the tampering is evident. The *core* encapsulates data operation interfaces with encrypted data, and isolates from the *shell*. In addition, we employ the *zero-knowledge proof* scheme to verify the request's identification without revealing any vital information which may be leveraged by hackers or attackers to disclosure user's data by forging a set of valid request parameters. Our active data-centric framework emphasizes proactivity and intelligence of data. We summarize our main contributions as follows:

- Our framework achieves platform-independent and multi-tenant support in the cloud. It does not rely on the third party trusted managers.
- Our security framework tightly binds to data. Verification and log of any access on data is enforceable, and this satisfies the requirement of accountability and auditability.
- Our framework achieves ubiquitous monitoring and tracking of data usage record, no matter where the data is located. Moreover, any access of data will be processed inside the data rather executed by a third party tool, guaranteeing minimum information disclosure of data.
- Our framework for data protection in the cloud is portable, active, and light-weight.
- Demonstration of the efficiency, dependability, and scalability of the framework with experiments on a real cloud test bed.
- We also provide a detailed analysis and evaluation of our scheme.

The rest of the paper is organized as follows. Section 2 introduces the related works in current research areas related with data protection and security in cloud computing. Section 3 presents an overall active data-centric framework and our previous works, demonstrates the framework infrastructure, and describes the corresponding core schemes. Section 4 presents the security analysis of our proposed schemes, followed by the performance study. Section 5 concludes the paper.

## RELATED WORKS

As cloud computing integrates and extends from existing Grid Technology, Service-Oriented Architecture, and Utility Computing Technology, issues that accompany these technologies are also retained in cloud computing. Conventional data protection schemes address security and privacy related issues through cryptographic primitives (Kallahalla et al. 2003; Yu et al. 2010) or data dispersal storage (Wang et al. 2009; Wang et al. 2011). These techniques adopted several effective and complicated key distribution or algorithms to guarantee data safety and integrity. However, they inevitably introduce complexity and unnecessary computational and communicational overhead. Besides that, these techniques do not provide support for accountability and auditability once user's data is mandated in the cloud side.

Several efforts (Abawajy 2011; Firdhous et al. 2011; Wang et al. 2010) introduced trust computing concept for cloud data security through a trusted third party management system. These schemes can efficiently increase the transparency of data usage and prevent their data from being compromised or leaked by the CSP. However, this type of solutions does not build in a protective capability on data once the adversary penetrates the data storage service and obtains direct access privilege on data, which is still under threat. Another representative approach is the data-centric protection which shifts protection mechanism from the third party protection environment to data where the data itself is strongly enforced according to the inner protection model that enables data to generate a self-defending and self-protection framework.

The work (Holford et al. 2004) designed a self-defending object which is an extension of the object oriented programming paradigm, and encapsulates sensitive data and functionality rather than being distributed throughout the application. This work was at an early-stage theoretic research, and no practical outcome was shown. Another similar work (Angin et al. 2010) proposed an entity-centric approach for privacy and identity management. It utilized active bundles and anonymous identification procedure to authenticate users to service providers to use the cloud service. This work has a similar concept to our work, but, differently, they encapsulated the personal identity information, privacy policy and a virtual machine in the entity, not data per se. Besides that, our research goal and techniques are entirely different.

Sundareswaran et al. (2011) and Sundareswaran et al. (2012) proposed a novel highly decentralized information accountability framework to keep track of data usage in the cloud. They leverage object-centred approach that encapsulates logging mechanism with users' data and policies to achieve a compulsory enforcement of auditing mechanism. However, our work is different from theirs on several aspects. Firstly, our TDFS supports a hierarchical data structure for controlling. Users can only access some portion of the data, not just support image-

type data. Secondly, we move our authorization and authentication work from inside the file to an external third party auditor due to several consideration: 1) our access control policy is designed to operate in the cloud context of highly distributed access control scenarios. Executing a fine-grained authentication in data file may cause more overhead to TDFS; 2) the policy is highly dynamic in the distributed cloud environment. The obsolete policy in TDFS may cause inconsistency of authentication. Moreover, we develop two level protection barriers to achieve tamper-proof scheme, which are the *shell* and the *core* in TDFS. We also employ the *zero-knowledge proof* scheme to verify the request identification without revealing any vital information which may be leveraged by hackers or attackers to disclosure data.

## ACTIVE DATA-CENTRIC FRAMEWORK

In the conventional computing-centric or application-centric data protection model, third party services emphasize an external security environment rather than on the data. In cloud computing environments, highly complex and dynamic hierarchical service chain determines that data handling may be delegated from a direct CSP to another different CSP in a flexible manner (Sundareswaran et al. 2011), and these CSPs do not enforce the same protection scheme and standard (Foster et al. 2008). Moreover, data may still be under threat and violation when adversaries penetrate into the cloud data storage layer even though we employ cryptographic schemes to increase the difficulty of revealing sensitive information. Furthermore, for accountability and auditability of data usage in the cloud, traditional probabilistic sampling technique (Wang et al. 2010) is not sufficient to provide fully accurate and timely auditing information due to its adoption of periodically fetching of data usage status. Based on these considerations, we propose an active data-centric framework that encloses an intelligent analysis mechanism along with users' data to generate an active object with self-defending and self-describing capabilities. Through the localization of intelligently active scripts within its encapsulating object, data's structure is enhanced. A violation on such data will automatically raise an alarm and alert its data owner or related legislation organization. The active data-centric framework facilitates data to defend external attack even though it is located in an untrusted environment. Comparing to computing-centric or application-centric data protection models, some verification and data operation functions in data-centric framework are executed by data rather than a third party service provider. Moreover, this framework can support a multi-tenant feature through independent and isolated data unit.

Our previous works (Chen and Hoang 2011) focused on implementing a robust data protection framework in a healthcare cloud, which is surrounded by a chain of protection schemes from designated access control, continuous monitoring, to active auditing from system-level to satisfy scalable, fine-grained, intrusion-tolerant, and consistent requirements. There are three crucial security components: Active Auditing Control (AAC) module, Cloud-based Privacy aware Role Based Access Control (CPRBAC) module, and Surveillance module in this service architecture. The AAC module offers a distributed transaction management scheme to strengthen security control integrity and keeps the auditing process from introducing new vulnerability of unauthorized information leakage towards the data security and privacy requirement. The CPRBAC module enables us to build designated cloud-based access control services that explicitly support highly dynamic distributed cloud environment and efficiently restrict users' access to cloud resources under fine-grained, complex context scenario. The surveillance module contains a group of verification monitors that are utilized to communicate with the embedded probes in our designated Triggerable Data File Structure (TDFS) in which we introduced its basic structure that consists of an executable segment, a header, data blocks and an encoding scheme. Relative to our previous work, this paper presents a novel active data-centric protection framework. The focus of this paper is to introduce and develop comprehensively our scheme from the active data-centric perspective. This includes enhanced construction of TDFS, transformation and trigger principle, tamper-proof scheme, and verification and identification of requests. Data encapsulates a number of runnable scripts, and we wrap the scripts and data with Java Archive (Jar) technology (Oracle 2011). The eventual active data can be run on the hosts which deployed a JVM environment. As the core of active data-centric framework, TDFS has an inner surveillance module, meanwhile, we propose a verification monitor which is an external monitor to incorporate with TDFS to maximize accountability and auditability of cloud data. Besides that, our TDFS is active on demand which means that it maintains an active status when it is triggered, and becomes static when it is unused or untouched. Making TDFS active all the time would cause unnecessary system overheads and may introduce access conflicts, and affect scalability requirement.

### TRIGGABLE DATA FILE STRUCTURE (TDFS)

As the core of our active data-centric framework, TDFS undertakes a major responsibility in protecting the inner data content from violation or attack, as well as providing efficient data provision service for incoming requests. In this section, we introduce our TDFS in details.

1. Construction of TDFS

As shown in figure 1, our proposed TDFS consists of a *shell* and a *core*. The *shell* is orchestrated with active tamper-proofing codes and is executed prior to the *core* when the TDFS is triggered. The TDFS is associated with a Java runtime environment. When it is triggered by third party entities, the *Java.exe* in JVM would invoke *GetMainClassName* function which obtains the *JNIEnv* instance, and then invokes *getManifest()* in *Java.util.jar.JarFileJNIEnv* class to return the object value which contains *Main-class*, the main class that is run when the TDFS is called. With that, the main function will invoke *LoadClass* method in *Java.c* to load the main class. Next, the main function will invoke the *GetStaticMethodID* method in *JNIEnv* instance to search for the "*public static void main (String[] args)*" method and then invoke the *CallStaticVoidMethod* method to process the incoming request.

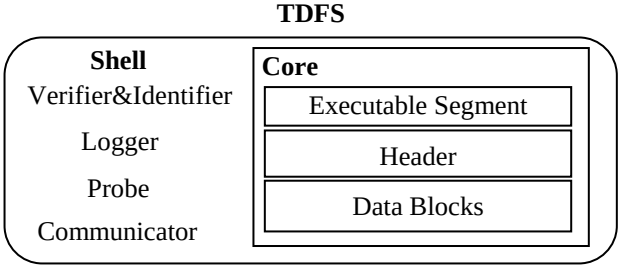


Figure 1: The skeleton of TDFS

On entering point of the *shell* the scripts invoke *verifier* and *identifier* to certify request's validity which includes the format of request parameters and contents through request identification and verification process as described in Section 4. In our context, the authentication and authorization related access control functions (Chen and Hoang 2011) are handled on externally designated servers. A permitted access request which is consistent with our pre-defined CPRBAC policy will be issued a certification that signifies that the current access request is derived from our AAC servers. The TDFS will then leverage the zero-knowledge proof scheme to verify and identify the request. This process will be discussed in the section 4.

Another significant component of *shell* is the *logger* module which is utilized to record significant checkpoints during the whole transaction, data operation outcomes, or even errors when TDFS throws exceptions. The *logger* is enforced to record significant inter-mediate information when the Jar is running in the JVM, which efficiently improves data accountability and auditability. Each log record in TDFS is in the form of {*Subject_ID, TDFS_ID, Entity, Behavior, TimeStamp, Location, Priority*} which indicates that a subject identified by *Subject_ID* performed a *Behavior* on TDFS identified by *TDFS_ID* with *Priority* level at *TimeStamp* at *Location*. During a singular transaction, all log records marked with regular *Priority* level are stored temporarily in memory for performance consideration. Once the data operation finishes, the *logger* leverages a *communicator* in the *shell* to upload the log records to the TDFS's external verification monitor. However, the log record marked with emergency tag will be immediately triggered by the *probe*, and then notify the *communicator* to raise an exception. *TimeStamp* uses the Network Time Protocol (NTP) to take into account the fact that cloud resources may distribute across different time zones. *Location* of TDFS is determined by IP address and MAC address of the current host. Each TDFS's log information is transparent to its data owner. When the log records are stored in the cloud, they are encrypted using the Public Key Infrastructure (PKI) to avoid possible information leakage. Only the data owner has the corresponding key to disclose them. In addition, sending out the log information of data usage rather than storing it inside the TDFS is considered for maintaining light-weight feature. The increasing log information may raise the cost of storage, transfer, or copy of TDFS.

The *communicator* uses RMI-SSL technique (Konstantinou 2003) to enhance the communication security. Each TDFS in the cloud has a corresponding supervisor which is a verification monitor in same domain level. These monitors take charge of monitoring external data operations (such as move, and copy, etc.) which cannot be detected by the internal *probe* in TDFS, and communicate with their TDFS. If the TDFS cannot establish a proper network connection or cannot contact its supervisor, it will switch to the termination state to avoid offline attack.

The *probe* in the *shell* is triggered by three types of activities: program exception, inconsistent checksum with data blocks, and verification failure of zero-knowledge proof procedure.

Once the verification and identification procedure succeeds, the *shell* will delegate the control to data *core*. The *core* of the TDFS is wrapped by an executable segment (ES), a header, and data blocks. The runnable scripts in the ES consist of some basic data operations, data loading, and data analysis functions. These functions are tightly coupled with the data. We leverage dynamic delegation approach in the ES to call back the *shell* to trigger the *core* and execute the data operations. The header refers to a manifest specifying some basic information of supplemental data resided at the beginning of a block of data. Data blocks describe our data with a tree-type structure which is a way of representing the hierarchical nature for a set of linked nodes. This tree-type data structure supports simultaneous data updates and deletes. Mathematically, it is an ordered directed tree separating

data contents into some organized and hierarchical sub-contents. Moreover, an encoding scheme on tree nodes is proposed in our previous work (Chen and Hoang 2011) to facilitate a quick determination of ancestor-descendent and parent-child relationships among nodes, querying and updating order-sensitive nodes in high efficiency. Our data value in the leaf-node is encrypted by system's public key, only the authorized users in our cloud service can obtain corresponding private key to decrypt the data blocks. Besides that, our data blocks also support other data formats, such as image, or media-stream.

## 2. Transformation process of TDFS

Our raw data is a static entity. Prior to transferring to cloud storage servers, the data is required to be transformed into a TDFS structure. Essentially, we leverage some active programmable technology such as Jar, etc. to wrap our data with java bytecode scripts which can execute on a JVM to form an active data unit. These embedded scripts take charge of request verification and identification, monitoring, and auditing tasks when the corresponding TDFS is triggered. Meanwhile, our data blocks should be reassembled and encrypted by system's public key. Table 1 gives description of notations that are used in our scheme. Figure 2 describes the transformation procedure.

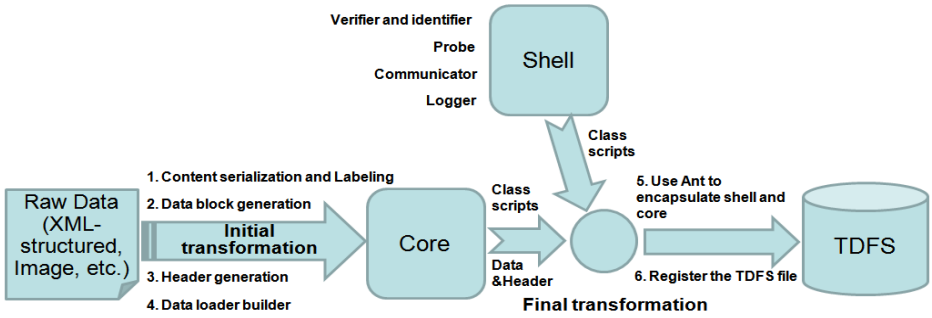


Figure 2: The transformation procedure from raw data to TDFS

This transformation procedure comprises of two stages: initial transformation and final transformation. Four steps are required to generate the *core* in the initial transformation: 1) Content serialization and labelling: firstly, data is required to be serialized to our system object in order to obtain attribute label set ($D_{set}$) for data blocks. *Raw data (RD)* of cloud users such as electronic health record in the healthcare area normally exists in the form of XML-type sequential structure specified by healthcare standards such as HL7 (Dolin et al. 2001). 2) Data block generation: A *PK* and *SK* key pair is created for the TDFS, *PK* is used to encrypt the $L_{set}$, and the corresponding *SK* is stored in the AAC server. The data block is then generated with $D_{set}$ and encrypted $L_{set}$. 3) Header generation: the $H_{set}$ defined in the header contains the generated manifest specifying the fundamental information of supplemental data. 4) Data loader builder: this step compiles the $JC_{core}$ and wraps them into the *core*. The final transformation utilizes Ant technique to encapsulate the *shell* and the *core* into a TDFS. The *shell* comprises of runnable scripts that include verifier and identifier module, probe, communicator, and logger. These modules are compiled into bytecodes which can be run on the JVM. After the completion of encapsulation of TDFS, it is registered with our cloud service.

Table 1. Notation used in the scheme description

| Notation | Description | Notation | Description |
|---|---|---|---|
| *PK, SK* | system public key and user security key | $JC_{shell}$ | all java codes for shell |
| $H_{set}$ | attribute set of header | $JC_{core}$ | all java codes for core |
| $D_{set}$ | attribute set of data blocks | *RD* | the raw data |
| $L_{set}$ | attribute value set of data blocks | | |

## 3. Tamper-proof scheme

There are many situations where attackers may utilize non-conventional approaches such as decompiling or reverse engineering to crack our TDFS internal defence to maliciously execute our TDFS. Theoretically, we cannot fully resist the adversaries from disclosure our data in cloud (Sundareswaran et al. 2012). They may leverage some vulnerabilities of system such as analysing compiled bytecodes, memory breakpoints to penetrate defence of our TDFS. However, what we can do is to increase the difficulty of disclosure data through non-symmetric encryption of the attribute value, enforceable monitoring and logging, or raising an alarm on the occurred violation, rather than prevent the occurrence of the attacks. To achieve these goals, our approach is to append an additional *shell* resided outside of data *core*. This concept is inspired by armored viruses (Guide 2011). The virus programmers create a "shell" program to protect or hide virus's content from disclosure to

escape the defence of anti-virus software. Although this may bring destructive effects on system or data because the virus could succeed escaping antivirus software via armored technique, we can leverage the technique to protect our TDFS. We define this layer's program as "shell". Our *shell* performs detection of tampering, verification and identification of request, remote communication, real-time logging, and terminating the access when tampering is evident. All components associated with data are wrapped into a *core* in TDFS. Sensitive data blocks are encrypted by the system's public key. They can be only revealed when the adversaries obtain the corresponding private key, while the private key normally resides in the highly trusted servers. Since we cannot guarantee that an adversary would not gain possession of our TDFS. To prevent TDFS from tampering attacks when the adversary penetrates our cloud security firewall (CPRBAC and AAC service) and obtain direct access of our TDFS, on the one hand, we have to rely on the strength of cryptographic primitives to guarantee the integrity and confidentiality of the data; on the other hand, the *shell* actually conceals the real entering point of data operations. It executes prior to the *core* in TDFS and the *core* only accepts requests from the *shell*. Hence, when the adversary attempts to extract useful information from our *shell* scripts to open the path to the *core*, we deploy the following approaches to detect the tampering:

- Verification and identification: We use the zero-knowledge proof scheme to verify the requester's identity through a three-pass protocol, which is described in section 4 in details.
- Integrity check: Before execution the verification and identification, the TDFS starts a self-check program to check the checksum of all bytecodes of *shell* and *core*. If the new checksum is inconsistent with the checksum stored in the *header*, the TDFS terminates the program immediately to block all requests, meanwhile, sends an alert message to its supervisor (verification monitor).
- Intermediate result check: We define checkpoints in whole program. When these checkpoints return abnormal data, TDFS will throw an exception and trigger the probe to raise an alarm. We can leverage this check to detect malicious code insertion.

4. Request identification and verification

This active data-centric approach adopts an executable framework to enforce data protection on an independent data unit basis. One of the crucial functions of the *shell* is to ascertain the validity of the incoming requests. Our goal is to allow the TDFS to verify the request identification without revealing any vital information which may be used by hackers or attackers to disclosure user's data by forging a set of valid request parameters. We describe the zero-knowledge proof scheme and the corresponding analysis in the following section.

**Zero-knowledge proof scheme:**

When the data request transaction arrives at the final commit stage, the AAC module will directly commit the transaction through triggering the requested TDFS with a valid proof which is normally issued by the AAC module to execute a data operation. The enforcement engine in TDFS requires a proof of authentication from the third party auditors. However, we assume that an active eavesdropper C has already penetrated our security system and obtained a straightforward access privilege on TDFS. C may still be capable of analysing the triggering parameters from the AAC to the TDFS, thereby forging a valid request even though C did not participate in the entire protocol. Hence, to distinguish a valid access from an invalid access (forged one) on the same data becomes a challenge for the TDFS. In general, the third party proof scheme strongly relies on network communication in the real world. Hence, the TDFS cannot fulfil the verification process once it cannot contact the third party trusted service. Moreover, this multi-parties communication procedure is easy to reveal the vulnerability to the adversaries who may execute a man-in-middle attack. But the zero-knowledge proof scheme (Pieprzyk et al. 2003; Schnorr 1990) only occurs between two entities. It is still able to provide the proof function even though the TDFS moves to an untrusted host without network connection. From the perspective of cloud computing architecture, data may be distributed across multiple domains geographically. This scheme is suitable to assist the TDFS to distinguish valid request and invalid request. Moreover, it can efficiently reduce the probability of forging of the proof due to zero information disclosure during the proof process. The following proposed scheme which is adapted with Schnorr identification scheme (Schnorr 1990):

1. Parameters initialization in the active auditing control (AAC) module:

1) AAC chooses primes $p$ and $q$ such that meet $p$-1 is divisible by $q$, and ensures the discrete logarithms of modules $p$ and $q$ are incalculable in theory. 2) Determines the security level through security parameter $t$, and meets $2^t < q$ . 3) AAC chooses $\beta$ with multiplication order $q$, and meets $1 \le \beta \le p-1$, $\beta^q \equiv 1 \bmod p$ . 4) Selects an one-way hash function $h(m)$ which calculates the signature for the message $m$. AAC publishes a set of security system parameters ($p$, $q$, $\beta$) and verification function $h(m)$.

2. Registration of cloud users:

1) When the request arrives at the AAC for registration the service, the AAC would verify the requester's identity, and then generates an identifiable string $I_A$ which contains the basic information about the user. 2) The user chooses a private key $a$ which meets $0 \le a \le q-1$ . AAC calculates corresponding public key

$v \equiv \beta^{-a} (\text{mod } p)$ . 3) The user verifies his identity from AAC through conventional presenting personal information, and then delivers the *v* to AAC. AAC eventually will issue a certification $cert_A = (I_A, v, h(I_A, v))$ which binds $I_A$ and *v.*

3. The identification protocol: (Entity A proves its identity to TDFS B)
1) *A* selects a random number *r* which meets $1 \le r \le q-1$, and computes $x \equiv \beta^r (\text{mod } p)$. *A* sends $(cert_A, x)$ to *B*. 2) *B* verifies the certification from *A* through the public key *v* in $cert_A$ in AAC, and then sends a random number *e* (which is never used) which meets $1 \le e \le 2^t$ to *A*. 3) *A* sends the answer $y \equiv a \cdot e + r (\text{mod } q)$ to *B*. 4) *B* computes $z \equiv \beta^y \cdot v^e (\text{mod } p)$ which can be decomposed as $z \equiv (\beta^y \bmod p * v^e \bmod p) \bmod p$, *B* accepts *A*'s proof if *z*=*x*, or refuse the *A*'s proof.

**Scheme Analysis:**
The AAC module provides the system security parameters. The protocol in this scheme takes three-round communications between the AAC module and the TDFS. Clearly, if A follows the protocol, it will be always correctly identified by our TDFS B. Now we analyse the security of the protocol from the following aspects:
1) Forge probability: we assume that the attacker *C* would like to guess *B*'s question, and let C's guess be g, *B* sends a question e and *A* has to respond $y \equiv a \cdot (e-g) + r (\text{mod } q)$ . As C is able to guess a valid response $y \equiv r (\text{mod } q)$, hence, the probability of correct guess of e is $2^{-t}$. If t $\ge$40, the success of probability will be less than $9 \cdot 10^{-13}$.
2) This scheme does not reveal any useful information related with *a*, since *x* is a random number, *y* requires *a* to calculate, *C* cannot answer *y* without *a*.
3) Discrete logarithm issue is difficult to compute in mathematic theory (Pieprzyk et al. 2003).
4) Once the verification time exceeds the normal time threshold, the TDFS would automatically terminate the verification procedure.
5) Once the private key *a* has been chosen, it is easy to compute the corresponding public key *v*. However, the inverse process, computing *a* from *v* requires to compute the discrete logarithm with base $\beta$ of $v^{-1}$.

## ANALYSIS OF OUR PROPOSED SCHEME

Cloud computing paradigm presents similar threats and issues to security and privacy as in traditional computing infrastructure. These issues mainly involve communication and network security in the internet/intranet/extranet environment (Krutz and Vines 2010). These traditional threats may come from within or external environments. The internal threats include eavesdropping of data, traffic or trend analysis, data manipulation, altering of data integrity, and the theft of information or unauthorized disclosure or operation of data. The external threats include diverse attacks such as distributed denial-of-service (DDoS) attacks, malicious intrusion or insertion of a malicious code or program. Additionally, in the traditional computing paradigm, users normally possess and utilize data in their local environment. In traditional scenarios the users take the responsibility of their data's safety, but in cloud environments new challenges are encountered around the data itself. Once the data is in the hand of the cloud service providers, the questions of "who, what, where, when, how and why" related to the user's data must be addressed as the cloud paradigm shifts user's responsibility from the user-side to the cloud-side. These new issues have to be addressed through new measures for providing sufficient transparency and accountability of users' data usage. In this section, we analyse possible attacks to our active data-centric framework in the cloud environment. These analyses are based on following assumptions: we assume that AAC service and verification monitor in cloud are sufficiently trustworthy, and they can provide correct legislation authentication for data requests. We assume that the RMI-SSL technique sufficiently strong to guarantee that the communication message between the two entities is secure. We assume that the runtime environment of our TDFS is not corrupted.

**Intrusion attack:** the most direct attack is that the adversary tries to reveal the contents of the TDFS through loading data input and output stream, and then runs it on corresponding runtime environment. Our scheme operates as follows. Any operations on TDFS will compulsorily trigger its shell protection scheme to execute verification and identification. Only the entity that passed the verification is allowed to continue the data access. A request without any parameter is straightway regarded as intrusion. The probe in TDFS would alert the intrusion event to external verification monitor to report an intrusion attack. If the adversary happens to know the request parameter structure, and enable to generate a correct data access parameter set, the TDFS would execute a verification and identification in terms of these parameters through a two rounds zero-knowledge proof scheme. Based on our previous assumption, the third party entities such as the AAC service and the verification monitor would not reveal any security-related information, that means only authorized entity issued by the AAC service or the verification monitor survives the verification and identification process without being noticed by our surveillance module in TDFS. Furthermore, breaking through our zero-knowledge proof scheme is equivalent to finding a polynomial-time algorithm to solve the discrete logarithm which is nearly impossible to achieve with

current computer technology (Pieprzyk et al. 2003). Once the verification time exceeds the normal time threshold, TDFS would proactively reject the request.

**Man-in-the-middle attack:** the adversary may intercept communication messages between the TDFS and its corresponding AAC service provider or verification monitor, and attempt to make them believe that they are communicating with each other over a correct connection. However, in fact the adversary may control the entire communication. This type of attacks basically can be prevented by deploying the RMI-SSL communication protocol which guarantees both entities using a mutually trusted communication tunnel, and we employ zero-knowledge proof scheme between the AAC service provider and the TDFS to avoid a third party from interruption.

**Malicious code insertion attack:** it is difficult to prevent an adversary to disassemble the TDFS files with abnormal approaches such as reverse-engineering, analyzing compiled bytecodes, memory breakpoints, etc. But, once the TDFS files are disassembled, the adversary may insert executable codes into the TDFS to break the shell protection scheme. This attack can be prevented by the sealing techniques. Sealing creates a signature that ensures the executable scripts inside the TDFS remains intact. Furthermore, integrity check and intermediate result check ensure that any malicious code insertion would be detected when the TDFS is running.

**TDFS external attacks:** if an adversary penetrates our cloud firewall and security system, and obtains a root privilege, the adversary is able to execute some operation system (OS) level operations on the TDFS, such as copy, move, and delete the entire TDFS from our system. These OS operations would not trigger the TDFS files into an active state, and hence the TDFS files may not be able to protect themselves. The solution we proposed is to deploy a verification monitor in the same domain as the TDFS files to take charge of an overall surveillance task. Once the above operations occur, the verification monitor can detect them from an OS probe, and then trigger the TDFS files into an active state. When TDFS files are active, they can execute self-protection process.
In brief, such an active data-centric framework dictates data's compulsory enforcement of verification and identification when it is triggered. In this framework, logging, verification, monitoring, and data operations are actively enforced along with the data. With the active data-centric framework, comprehensive transparency can be achieved through data usage monitoring and protection. For cloud users, once their data is compromised or violated in their subscribed cloud service, cloud service providers can detect violations and inform their customers through a bundled mobile phone push service. Furthermore, any illegal violations could be delivered to relevant governance, regulation, and compliance authorities.

## PERFORMANCE STUDY AND IMPLEMENTATION

Our experiments were performed on the IBM HS20 Blades center which consists of 8 high-performance blade servers. Each server is equipped with an Intel Xeon CPU 2.8 GHZ, 2GB RAM, and connects to a DS400 SAN (Storage Area Networking) by a 10G Fiber-Channel switch. We selected 3 of 8 Blades as our mini-private cloud infrastructure. These three blade servers separately install Ubuntu-10.04.2-desktop-i386 OS, and we utilized OpenStack open source cloud toolkit (OpenStack 2011) to establish and manage our cloud platform resources. In this research, we do not focus on the construction of the cloud platform, rather our focus is on the security and privacy issues around cloud environments. In the experiments, we first examine the time taken to retrieve data from the TDFS, and then test several invalid intrusion cases simulated through a third party program which is developed in Java, finally, we measure the storage cost and transformation cost in our active data-centric framework. Our static data derived from (Dolin et al. 2000), with only small parts of the EHR data were captured. The data blocks in this test case consist of 15 content nodes. The data complied with HL7 standard.

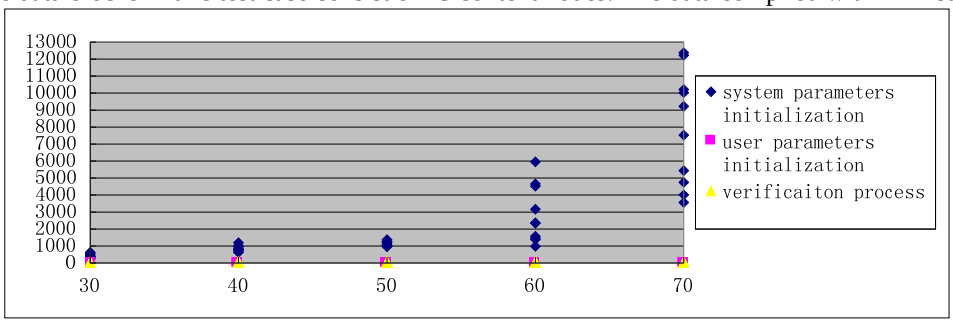


Figure 3: Time cost of zero-knowledge proof scheme from TDFS

1. Time cost for a complete data retrieval from a TDFS: The overall overhead for a complete data retrieval from a TDFS involves verification and identification, data loading, and network communication cost. Basically, the network communication cost is negligible due to our high speed intranet test environment. We mainly tested the performance of zero-knowledge proof scheme and data loading. The most important element which decides the performance in the stage of verification is the $q$'s bit length. Meanwhile $q$ also determines $p$, and $t$. Due to great

randomness of generating an appropriate prime number with certain bit lengths and meeting *p-1* is divisible by *q*, the time cost for the parameters initialization in the AAC module demonstrates obvious variation. Hence, we did 5 sets of tests in which the bit length for the *q* is ranged from 30 to 70, and each set of test runs 10 times circularly. The figure 3 illustrates the test result. The x-axis represents the bit length of *q*, and the y-axis represents the execution time. As we can see, along with the bit lengths of *q* increasing, the time cost for generating the system public parameters increases obviously. However, the average time cost for 40 bit length's *q* (844.9ms) is similar to 50 bit length's *q* (1126.1ms). Considering the time taken for generating the system public parameters may be a bottleneck for achieving the lightweight feature of TDFS, we suggest that the bit length of *q* can be from 40 bits to 50 bits. In that way, the *q* will be larger than $2^{130}$, and *t* can be larger than 65, hence the success of probability for adversaries guessing the correct *e* will be less than $2.7 \cdot 10^{-20}$, which is sufficiently secure to resist the discrete logarithm attack. In addition, the time cost for user parameters generation and verification process nearly maintain constant which is around 50ms. The time cost for data loading in this test case is around 170ms.

2. Test cases for intrusion attacks: To test the alert functions for the intrusion attacks, we developed a third party program to simulate a simple intrusion attack scenario. To implement violation alert function, we developed the notification feature of the Cloud to Device Messaging Framework (Android C2DM) in our platform. It binds with cloud user's data violation. We established three test cases: the first one is triggering the TDFS file without any parameters; the second one is moving the TDFS without any permission from AAC module; and the third one is triggering the TDFS file with correct parameters structure. All these three test cases triggered the *probe* inside the TDFS because they failed at the verification stage and then violation messages are sent to our verification monitor, and meanwhile, our mobile device received the alert message immediately. Figure 4 demonstrates the screenshots of the notification messages in Samsung Galaxy SII when the violations occur in user's TDFS file.

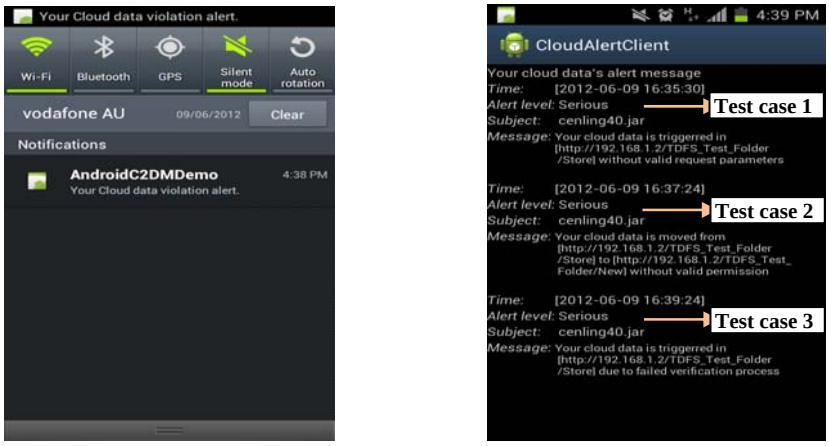


Figure 4: Screenshots of notification messages when executing the three intrusion test cases

3. Storage cost and transformation cost: Finally, we investigated the shell size, core size, and final TDFS size. Basically, the shell occupies 333KB storage and the core occupies 90.8KB in which the data blocks, header and the image data occupy 64.9KB. However, the final TDFS only occupies 250KB storage due to the compression feature of Jar technology. The size of TDFS will vary only when the size of the data blocks and header change and the other runnable scripts in shell and core are unchangeable. To test average transformation cost, we leverage the "ant" technique (Kallambella 2006) to automatically transform the raw EHR data into the TDFS. The average transformation time was around 200 ms when executed 30 times circularly. The overall outcome also presents a light-weight feature.

## CONCLUSION

This paper presented a novel active data-centric framework for data protection in cloud scenario. It provided higher transparency and accountability to efficiently mitigate users' concern about the protection of their data in outsourced cloud environments. We proposed a novel TDFS in which the data is equipped with the intelligent active properties to achieve self-describing and self-defending capabilities in a light-weight protection framework via an active bundle scheme. The scheme strongly enforces data protection requirements and strictly conforms to high transparency of data usage. Moreover, we implemented the zero-knowledge proof scheme in TDFS to verify the request's identification without revealing any vital information about the data.

## REFERENCES

Abawajy, J. 2011. "Establishing Trust in Hybrid Cloud Computing Environments," *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 118-125.

Angin, P., Bhargava, B., Ranchal, R., Singh, N., Linderman, M., Ben Othmane, L., and Lilien, L. 2010. "An Entity-Centric Approach for Privacy and Identity Management in Cloud Computing," *29th IEEE Symposium on Reliable Distributed Systems*, pp. 177-183.

Chen, L., and Hoang, D.B. 2011. "Towards Scalable, Fine-Grained, Intrusion-Tolerant Data Protection Models for Healthcare Cloud," *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 126-133.

Dolin, R.H., Alschuler, L., Beebe, C., Biron, P.V., Boyer, S.L., Essin, D., Kimber, E., Lincoln, T., and Mattison, J.E. 2001. "The Hl7 Clinical Document Architecture," *Journal of the American Medical Informatics Association* (8:6), p 552.

Dolin, R.H., Alschuler, L., Boyer, S., and Beebe, C. 2000. "An Update on Hl7's Xml-Based Document Representation Standards," American Medical Informatics Association.

Firdhous, M., Ghazali, O., and Hassan, S. 2011. "A Trust Computing Mechanism for Cloud Computing with Multilevel Thresholding," *6th IEEE International Conference on Industrial and Information Systems (ICIIS)*, pp. 457-461.

Foster, I., Yong, Z., Raicu, I., and Lu, S. 2008. "Cloud Computing and Grid Computing 360-Degree Compared," *Grid Computing Environments Workshop, 2008. GCE '08*, pp. 1-10.

Goh, E.J., Shacham, H., Modadugu, N., and Boneh, D. 2003. "Sirius: Securing Remote Untrusted Storage," Citeseer, pp. 131–145.

Goyal, P., and Mikkilineni, R. 2009. "Policy-Based Event-Driven Services-Oriented Architecture for Cloud Services Operation Management," *IEEE International Conference on Cloud Computing (CLOUD)*, pp. 135-138.

Grove, D. 2011. "Healthcare Practices Embrace Mobile Technologies, New Comptia Research Reveals " Retrieved December 13th, 2011, from http://www.comptia.org/news/pressreleases/11-11-16/Healthcare_Practices_Embrace_Mobile_Technologies_New_CompTIA_Research_Reveals.aspx

Guide, I. 2011. "Computer Viruses / Virus Guide." from http://www.internet-guide.co.uk/viruses.html

Holford, J.W., Caelli, W.J., and Rhodes, A.W. 2004. "Using Self-Defending Objects to Develop Security Aware Applications in Java\&Trade," *Proceedings of the 27th Australasian conference on Computer science - Volume 26*, Dunedin, New Zealand: Australian Computer Society, Inc., pp. 341-349.

Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., and Fu, K. 2003. "Plutus: Scalable Secure File Sharing on Untrusted Storage," *03 Proceedings of the 2nd USENIX Conference on File and Storage Technologies* pp. 29–42.

Kallambella, A. 2006. "Advanced Ant Techniques, Part I." Retrieved September, 2011, from http://www.javaranch.com/journal/200603/AntPart1.html

Ko, R.K.L., Jagadpramana, P., and Bu Sung, L. 2011a. "Flogger: A File-Centric Logger for Monitoring File Access and Transfers within Cloud Computing Environments," *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 765-771.

Ko, R.K.L., Lee, B.S., and Pearson, S. 2011b. "Towards Achieving Accountability, Auditability and Trust in Cloud Computing," A. Abraham, J.L. Mauri, J.F. Buford, J. Suzuki and S.M. Thampi (eds.). Springer Berlin Heidelberg, pp. 432-444.

Konstantinou, A.V. 2003. "Using Rmi over Ssl Authentication for Application-Level Access Control." from http://www.cs.columbia.edu/~akonstan/rmi-ssl/

Krutz, R.L., and Vines, R.D. 2010. *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*.

Lin, D., and Squicciarini, A. 2010. "Data Protection Models for Service Provisioning in the Cloud," *15th ACM symposium on Access control models and technologies*, Pittsburgh, Pennsylvania, USA: ACM, pp. 183-192.

NTP. "The Network Time Protocol." from http://www.ntp.org/

OpenStack. 2011. "Open Source Software for Building Private and Public Clouds." Retrieved October, 2011, from http://openstack.org/

Oracle. 2011. "The Java Archive (Jar) File Format Using Jar Files: The Basics." Retrieved September, 2011, from http://java.sun.com/developer/Books/javaprogramming/JAR/basics/

Pieprzyk, J., Hardjono, T., and Seberry, J. 2003. *Fundamentals of Computer Security*. Springer.

Schnorr, C. 1990. "Efficient Identification and Signatures for Smart Cards
Advances in Cryptology — Crypto' 89 Proceedings," G. Brassard (ed.). Springer Berlin / Heidelberg, pp. 239-252.

Sundareswaran, S., Squicciarini, A., Dan, L., and Shuo, H. 2011. "Promoting Distributed Accountability in the Cloud," *2011 IEEE International Conference on Cloud Computing (CLOUD)*, pp. 113-120.

Sundareswaran, S., Squicciarini, A., and Lin, D. 2012. "Ensuring Distributed Accountability for Data Sharing in the Cloud," *IEEE Transactions on Dependable and Secure Computing* (9:4), pp 556-568.

Takabi, H., and Joshi, J.B.D. 2012. "Policy Management as a Service: An Approach to Manage Policy Heterogeneity in Cloud Computing Environment," *45th Hawaii International Conference on System Science (HICSS)*, pp. 5500-5508.

Wang, C., Wang, Q., Ren, K., and Lou, W. 2009. "Ensuring Data Storage Security in Cloud Computing," *17th International Workshop on Quality of Service (IWQoS)* pp. 1-9.

Wang, C., Wang, Q., Ren, K., and Lou, W. 2010. "Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing," *2010 IEEE Proceedings on INFOCOM*, pp. 1-9.

Wang, S., Agrawal, D., and El Abbadi, A. 2011. "A Comprehensive Framework for Secure Query Processing on Relational Data in the Cloud," *8th VLDB international conference on Secure data management*, Seattle, WA, pp. 52-69.

Yu, S., Wang, C., Ren, K., and Lou, W. 2010. "Achieving Secure, Scalable, and Fine-Grained Data Access Control in Cloud Computing," *INFOCOM'10 Proceedings of the 29th conference on Information communications*, pp. 534-542

## COPYRIGHT