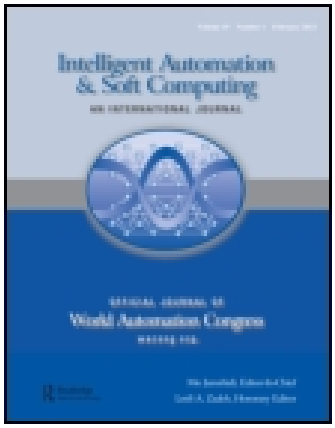

Intelligent Automation & Soft Computing
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tasj20

Cryptographic Cloud Storage with Public Verifiability: Ensuring Data Security of the YML Framework

Xin Lv [a b], Feng Xu [a] & Serge G. Petiton [b]
[a] College of Computer and Information, Hohai University, Nanjing, P.R. China
[b] Laboratoire d'Informatique Fondamentale de Lille, University of Sciences and Technology of Lille, Lille, France
Published online: 17 Apr 2013.

To cite this article: Xin Lv, Feng Xu & Serge G. Petiton (2013) Cryptographic Cloud Storage with Public Verifiability: Ensuring Data Security of the YML Framework, Intelligent Automation & Soft Computing, 19:2, 111-121, DOI: 10.1080/10798587.2013.786958

To link to this article: http://dx.doi.org/10.1080/10798587.2013.786958

Intelligent Automation and Soft Computing, 2013
Vol. 19, No. 2, 111–121, http://dx.doi.org/10.1080/10798587.2013.786958

CRYPTOGRAPHIC CLOUD STORAGE WITH PUBLIC VERIFIABILITY: ENSURING DATA SECURITY OF THE YML FRAMEWORK

Xin Lv[1,2], Feng Xu[1]* and Serge G. Petiton[2]
[1]*College of Computer and Information, Hohai University, Nanjing, P.R. China*
[2]*MAP Team, Laboratoire d'Informatique Fondamentale de Lille, University of Sciences and Technology of Lille, Lille, France*

ABSTRACT—YML framework is a well-adapted advanced tool to support designing and executing portable parallel applications over large scale peer to peer and grid middlewares. This work studies the problem of ensuring data security of the YML framework. We define and construct a mechanism that enables us to move the data repository to a public cloud infrastructure where the service provider is not completely trustworthy. To achieve confidentiality, we encrypt the data using the encryption algorithm in our prior work before uploading to the cloud, and then attach pre-classified keywords to them for ciphertext-searching, which are generated by a statistically consistent public-key encryption with keyword search (PEKS) scheme, so the service provider can use the corresponding trapdoor to identify all data containing some specific keywords without learning anything else. To ensure integrity, an elegant verification scheme is proposed, enabling a third party auditor (TPA), on behalf of data owner, to verify the integrity of the (encrypted) data stored in the cloud. The introduction of TPA eliminates the involvement of client through the auditing of whether his data stored in the cloud is indeed intact, which can be important in achieving economies of scale for cloud computing.

Key Words: Data Repository; Cryptographic Cloud Storage; Bilinear Diffie-Hellman Problem; Ciphertext-Searching; Statistical Consistency; Public Verifiability

1. INTRODUCTION

YML [1] framework has been developed since 2000. It acts as a well-adapted advanced tool to support designing and executing portable parallel applications over large scale peer to peer and grid middlewares [2,3]. YML includes a workflow language named *YvetteML* used in the description of applications and their executions. YML furnishes a compiler and a just-in-time scheduler for *YvetteML*. It allows the user to manage the execution of the application over the underlying parallel architecture which can be a peer to peer or a grid middleware. The specificity of each middleware is hidden to the user through YML, making the user can easily develop a complex parallel application which may transparently execute on multiple middlewares during one application execution. The framework provides workflow engine capabilities on top of a global computing platform, and it is designed to act transparently for complex applications using numerous communications, code coupling, etc. on dynamic platforms. The extension version of YML is able to manage at the run-time several middleware back-ends, achieving a dynamic federation of computing middleware [4]. We also extend the framework to be middleware for cloud computing platform [5,6].

YML framework implies a lot of data exchange through the network. The Data Repository server acts as a resource provider and delivers data to each component on demand. It is also the mediator for data exchanging of the peers. This fashion hides the data migrations to the developer and ensures that necessary

*Corresponding author. Email: njxufeng@163.com

data are always available to all components of the application, also making the Data Repository become a center which needs to be highly protected. In paper [7], we proposed an efficient threshold encryption scheme to safeguard the process of data exchanging between each party (data repository, other components, or peers). In this scheme, the data is transformed into *half-ciphertext* which is able to transfer in public channel securely, and the sender (receiver) can efficiently prepare (recover) half-ciphertext (plaintext) using his own piece of the key.

On the basis of this work, we are trying to move the Data Repository to a public cloud for reducing capital and operational expenditures. While the benefits of using a public cloud infrastructure are clear, it introduces significant security and privacy risks. In fact, it seems that the biggest hurdle to the adoption of cloud storage is concern over the confidentiality and integrity of data. In the view of these two key points, in this paper we present a cryptographic cloud storage scheme to ensure data security of the YML framework. In our scheme, the data is stored on the cloud in ciphertext for confidentiality, using the encryption scheme proposed in [7], particularly, we append to the ciphertext a public-key encryption with keyword search of each keyword. As to integrity, we equip the verification protocol with public verifiability, allowing anyone to have the capability to verify the correctness of the stored data on demand without the local copy of data files, that is to say, we are able to delegate the evaluation of the service performance to an independent third party auditor, without devotion of our own computation resources.

The rest of the paper is organized as follows. We explain the application scenario and the architecture of the proposed scheme in Section 2, including overview of YML, system model, and security model. A concrete construction is then given in Section 3. It is followed by a specific security and efficiency analysis on the concrete scheme in Section 4, 5. Finally we conclude the paper in Section 6.

2. PROBLEM STATEMENT

2.1 Overview of YML

YML is a framework dedicated to the creation and the execution of parallel and distributed applications on various middleware. It proposes an intuitive representation of a distributed and parallel application by means of a workflow. The main structures of YvetteML are: the services execution, the parallel sections, the sequential loops, the parallelized loops, the conditional branch and the event notification/ reception.

YML represents the notion of computing task by components, called services. Each computational task is described by an *abstract service* and is implemented in an *implementation service*. Services information is contained into two catalogs. A Development Catalog stores information used at the time of the application development. An Execution Catalog stores information used during the execution of the application.

Figure 1 gives a simplified view of an YML framework. We notice that there are four entities.

The CLIENT provides the computational components (abstract and implementation services) and the application graph expressing the control workflow.

YML workers are managed by any kind of middleware as long as a back-end enables to handle it.

YML is the core of the "sandwich" architecture. It hides to the client the complexity and the heterogeneity of the computing platform. YML is composed of a front-end layer and a back-end layer.

The Data Repository server interacts with the YML workers and the YML front-end. Basically, this server is in charge of storing the binary of components and the input parameters. It delivers such data to the YML workers and collects output results which are used by the YML scheduler to select the next eligible task.

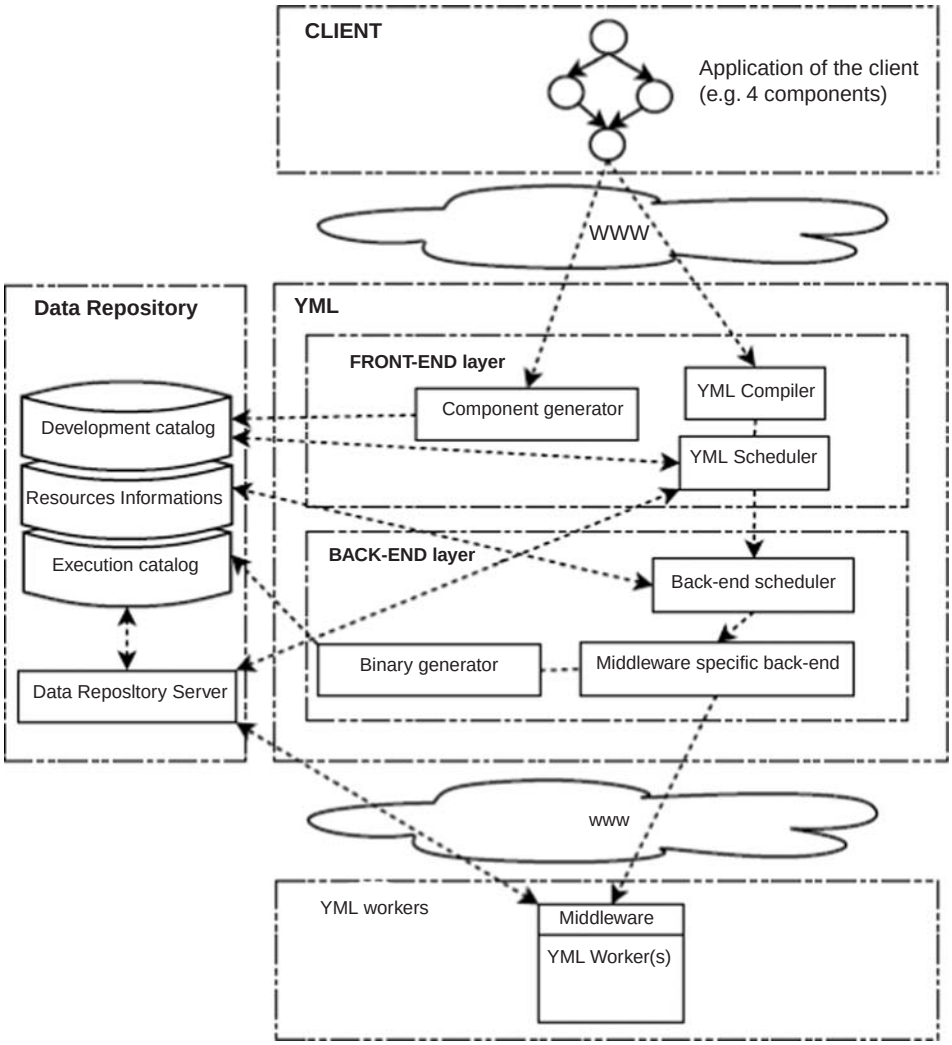

Figure 1. Overview of YML framework.

2.2 System model

A representative architecture for cryptographic cloud storage is illustrated in Figure 2. Five different entities can be identified as follow: three of them are same as that in the traditional YML framework (CLIENT, YML, YML workers), and Data Repository has been moved to a Cloud Storage Server, besides, Third Party Auditor (TPA) is in charge of verifying the integrity of the data.

In the cloud paradigm, by putting the large data files on the remote servers, we are relieved of the burden of storage and computation. As we no longer possess our data locally, it is necessary to ensure that the data are being correctly stored and maintained. In this paper, we delegate the monitoring task to a trusted TPA: any TPA in possession of the public key can act as a verifier (to protect data privacy, audits are performed without revealing original data files to TPA). We assume that TPA is unbiased while the server is untrusted.

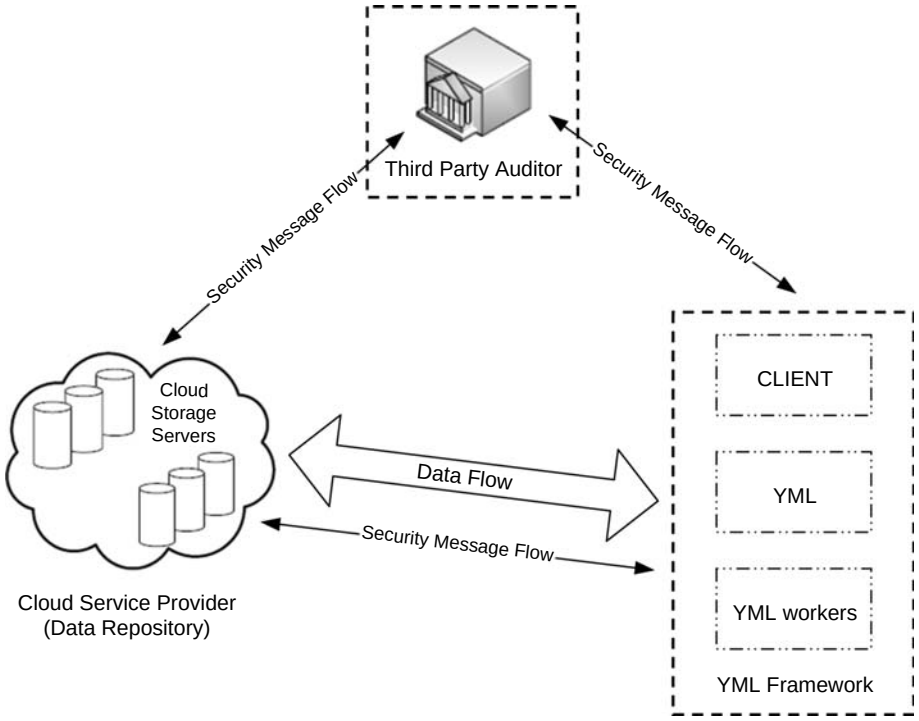

Figure 2. Cryptographic cloud storage architecture.

2.3 Security model

Our security model is the same as the one proposed in [8] for PoR system. Generally, the checking scheme is secure if (i) there exists no polynomial-time algorithm that can cheat the verifier with non-negligible probability; (ii) there exists a polynomial-time extractor that can recover the original data files by carrying out multiple challenges-responses. The authors in [8] also define the correctness and soundness of PoR scheme: the scheme is correct if the verification algorithm accepts when interacting with the valid prover (e.g., the server returns a valid response) and it is sound if any cheating server that convinces the client it is storing the data file is actually storing that file.

3. THE PROPOSED SCHEME

3.1 Notation and preliminaries

Bilinear Pairing. A bilinear pairing is a map $e{:}G_1 \times G_1 \rightarrow G_2$, where G_1 is an additive cyclic group of prime order p and G_2 is a multiplicative cyclic group with the same order. The map satisfies the following properties [9]: (i) Computable: given g, $h \in G_1$ there is a polynomial time algorithm to compute $e(g, h) \in G_2$; (ii) Bilinear: for any integers x, $y \in [1, p]$ we have $e(g^x, g^y) = e(g, g)^{xy}$; (iii) Non-degenerate: if g is a generator of G_1 then $e(g, g)$ is a generator of G_2.

Efficient Threshold Encryption Scheme [7]. The scheme we proposed in [7] is based on the RSA cryptosystem. (e, d) is the RSA public/secret key, and a trusted third party (TTP) divides d into enough pieces d_i using a simple polynomial $f(x)$, making any two pieces are matched. Then it distributes them to

each entity in the YML framework. In the process of encrypting and decrypting, the sender firstly encrypts the data with d, and then decrypts the ciphertext preliminarily with his own key (piece), finally sends the half-ciphertext to the receiver (transfers to data repository at first, and then it can be retrieved by the others). Because any two pieces can collaborate to complete decrypting, the receiver is able to obtain the data (plaintext) immediately. The following are the concrete steps:

(1) Entity i (sender) encrypts the data: $c = m^e \bmod N$;
(2) Then decrypts the ciphertext preliminarily: $c_h = (c)^{d_i \cdot (-x_j)/(x_i - x_j)} \bmod N$;
(3) Transfer (c, c_h) to the data repository, and then they can be retrieved by entity j;
(4) Entity j (receiver) obtain the data: $m = (c)^{d_j \cdot (-x_i)/(x_j - x_i)} \bmod N$.

We plan to store all the (c, c_h) on the public cloud instead of the Data Repository with pre-classified keywords, then the peers or the YML components can directly retrieve the data from it, so the efficiency of the data exchanging must be advanced.

3.2 Definitions

3.2.1 Public-key encryption with searching: definition

Before uploading the data to the cloud, we firstly transform them from plaintext to half-ciphertext, in order to make the resulting ciphertext searchable, a secure public key encryption with keyword search (PEKS) scheme is utilized. It appends the keywords to the ciphertext, so the encrypted data with keywords W_1, $W_2, \ldots, W_k$ is in the following form:

$$[D_{HC}, PEKS(D_{pub}, W_1), \ldots, PEKS(D_{pub}, W_k)]$$

where D_{HC} is the half-ciphertext of the data, D_{pub} is the public key of the Data Repository, and PEKS is an algorithm with properties discussed below. The PEKS values do not reveal any information about the data, but enable searching for specific keywords.

The goal is to enable the cloud server to locate all data containing the keyword W using a short secret key T_W generated by us, but learn nothing else. We produce this trapdoor T_W by the private key of the Data Repository. Then the server simply sends the relevant ciphertext back to the YML components or the peers. We call such a system non-interactive public key encryption with keyword search.

Definition 1. A non-interactive public key encryption with keyword search scheme consists of the following polynomial time randomized algorithms:

(1) $KeyGen(1^k)$: takes a security parameter k, and generates a public/private key pair D_{pub} and D_{priv}.
(2) $PEKS(D_{pub}, W)$: for a public key D_{pub} and a word W, produces a searchable encryption of W.
(3) $Trapdoor(D_{priv}, W)$: given the private key of the data repository and a word W, produces a trapdoor T_W.
(4) $Test(D_{pub}, S, T_W)$: given the public key of the data repository, a searchable encryption $S = PEKS(D_{pub}, W')$, and a trapdoor $T_W = Trapdoor(D_{priv}, W)$, outputs 'yes' if $W = W'$ and 'no' otherwise.

We run the *KeyGen* algorithm to generate the public/private key pair of the data repository. The data requester (peers or the YML components) uses *Trapdoor* to generate trapdoors T_W for any keywords W that it wants the cloud server to search for. The cloud server uses the given trapdoors as input to the *Test* algorithm to determine whether given data contains one of the keywords W specified by the data requester.

3.2.2 Public verification for storage correctness assurance

In order to make the outsourced data verifiable, we design a signature scheme, providing an efficient way, to allow anyone (TPA), not just ourselves, to perform periodical integrity verifications on demand.

Definition 2. An integrity assurance signature scheme consists of the following algorithms:

(1) *KeyGen*(1^k): this probabilistic algorithm is run by ourselves. It takes as input security parameter 1^k, and returns public key *pk* and private key *sk*.
(2) *SigGen*(*sk*, *D*): It takes as input private key *sk* and a file *D* which is an ordered collection of blocks $\{c_i\}$, and outputs the signature set Φ, which is an ordered collection of signatures $\{\sigma_i\}$ on $\{c_i\}$.
(3) *GenProof*(*D*, Φ, *chal*): this algorithm is run by the cloud server. It takes as input a file *D*, its signatures Φ, and a challenge *chal*. It outputs a data integrity proof *P* for the blocks specified by *chal*.
(4) *VerifyProof*(*pk*, *chal*, *P*): this algorithm can be run by either ourselves or the third party auditor upon receipt of the proof *P*. It takes as input the public key *pk*, the challenge *chal*, and the proof *P* returned from the server. It outputs *TRUE* if the integrity of the file is verified as correct or *FALSE* otherwise.

3.3 Construction

In our construction, we use BLS signature [10] as a basis to design the system. G_1 and G_2 are the groups defining in Section 3.1, and g is a generator of G_1. Let e: $G_1 \times G_1 \rightarrow G_2$ be a bilinear map, and H_1: $\{0, 1\}^* \rightarrow G_1$, H_2: $G_2 \rightarrow \{0, 1\}^{3k}$, H_3: $\{0, 1\}^* \rightarrow \{0, 1\}^k$, and H_4: $\{0, 1\}^* \rightarrow \{0, 1\}^k$ are random oracles. $f(k) = k^{\lg(k)}$. k is a security parameter. The procedure of our protocol execution is as follows:

■ *Pre-Processing:* we first process the data to half-ciphertext D_{HC} using the encryption scheme mentioned in Section 3.1, and then prepare the Public-Key Encryption with Keyword Search (PEKS) for each pre-classified keyword, using a scheme defined by Definition 1, which will be attached to the data before uploading to the cloud.

$KeyGen(1^k)$
$\alpha \leftarrow_R Z_p^*$; $D_{pub} \leftarrow (1^k, g, g^\alpha, G_1, G_2, p, e)$
$D_{priv} \leftarrow (D_{pub}, \alpha)$; return (D_{pub}, D_{priv}).
$PEKS^{H_1, H_2, H_3, H_4}(D_{pub}, W)$
if $|W| \geq f(k)$ then return W
$r \leftarrow_R Z_p^*$; $T \leftarrow e(g^\alpha, H_1(W))^r$
$K_1 \leftarrow H_4(T)$; $K_2 \leftarrow H_2(T)$
$K \leftarrow_R \{0, 1\}^k$; $c \leftarrow K_1 \oplus K$
$t \leftarrow H_3(K \| W)$
return (g^r, c, t, K_2)

■ *Signature Generation:* as in the previous PoR system [11], we encode the raw ciphertexts $\tilde{D} = \{D_C, D_{HC}\}$($D_C$, D_{HC} are the ciphertext and half-ciphertext of the data respectively) into D using Reed-Solomon codes and divide the encoded file D into n blocks $c_1, c_2, \ldots, c_n$, where $c_i \in Z_p^*$.

The public and private keys used in this process are generated by invoking *KeyGen*(1^k) defined in Definition 2. Here we use the same public key and secret key in *Pre-Processing*, that is, *pk* equals to D_{pub}, and *sk* equals to D_{priv}.

SigGen(*sk*, *D*)
parse *D* as $(c_1, c_2, \ldots, c_n)$
$u \leftarrow_R G_1$; $\sigma_i \leftarrow (H_1(c_i) \cdot u^{c_i})^{\alpha}$ for each $c_i (i = 1, 2, \ldots, n)$
return $\Phi = (\sigma_1, \sigma_2, \ldots, \sigma_n)$
We finally upload $\{[D, PEKS^{H_1, H_2, H_3, H_4}(D_{pub}, W)], \Phi\}$ to the cloud server and delete them from its local storage.

■ *Ciphertext-Searching:* when the data requester wants to retrieve the encrypted data tagged with a certain keyword, *Trapdoor*(D_{priv}, *W*) is invoked to create the corresponding trapdoor T_W. The trapdoor is sent to the cloud server who uses it to carry out *Test*(D_{pub}, *S*, T_W), retrieving the appropriate (encrypted) files which it returns to the data requester. Then it can decrypt the ciphertext using its own piece of the decryption key.

$Trapdoor^{H_1}(D_{priv}, W)$
$T_W \leftarrow (H_1(W)^{\alpha}, W)$
return T_W
$Test^{H_1, H_2, H_3, H_4}(D_{pub}, S, T_W)$
if $|W| \geq f(k)$ then
if $S = W$ then return 1 else return 0
if *S* cannot be parsed as (g^r, c, t, K_2) then return 0
$T \leftarrow e(g^r, H_1(W)^{\alpha})$
$K \leftarrow c \oplus H_4(T)$
if $K_2 \neq H_2(T)$ then return 0
if $t = H_3(K \| W)$ then return 1 else return 0

■ *Proof Generation:* we or the third party, e.g., TPA, can verify the integrity of the outsourced data by challenging the server. To generate the message "*chal*", the TPA (verifier) picks a random c-element subset $I = \{s_1, s_2, \ldots, s_c\}$ of the set $[1, n]$, where we assume $s_1 \leq s_2 \leq \cdots \leq s_n$. For each $i \in I$, the TPA chooses a random element $v_i \leftarrow Z_p^*$. The message "*chal*" specifies the positions of the blocks to be checked in this challenge phase. The verifier sends the *chal* $\{(i, v_i)\}_{s_1 \leq i \leq s_c}$ to the prover (server). Upon receiving the challenge, the server executes *GenProof* (*D*, Φ, *chal*) to generate the proof.

GenProof (*D*, Φ, *chal*)
parse *D* as $(c_1, c_2, \ldots, c_n)$
Φ as $(\sigma_1, \sigma_2, \ldots, \sigma_n)$
chal as $\{(i, v_i)\}_{s_1 \leq i \leq s_c}$
$\mu \leftarrow \sum_{i=s_1}^{s_c} v_i c_i \in Z_p^*$; $\sigma \leftarrow \prod_{i=s_1}^{s_c} \sigma_i^{v_i} \in G_1$
return $P = \{\mu, \sigma, H_1(c_i)_{s_1 \leq i \leq s_c}\}$
Then, the prover responds the verifier with proof $P = \{\mu, \sigma, H_1(c_i)_{s_1 \leq i \leq s_c}\}$.

■ *Proof Verification:* upon receiving the responses from the prover, the verifier runs *VerifyProof*(*pk*, *chal*, *P*) to verify the integrity of the selected blocks.

VerifyProof (*pk*, *chal*, *P*)
parse *chal* as $\{(i, v_i)\}_{s_1 \leq i \leq s_c}$
P as $\{\mu, \sigma, H_1(c_i)_{s_1 \leq i \leq s_c}\}$
if $e(\sigma, g) = e\left(\prod_{i=s_1}^{s_c} H_1(c_i)^{v_i} \cdot u^{\mu}, g^{\alpha}\right)$ then return *TRUE* else return *FALSE*

4. SECURITY ANALYSIS

Bilinear Diffie-Hellman Problem (BDH): g is a generator of G_1, given g^a, g^b, $g^c \in G_1$ as input, compute $e(g, g)^{abc} \in G_2$. We say that BDH is intractable if all polynomial time algorithms have a negligible advantage in solving BDH.

Privacy. Privacy for a PEKS scheme asks that an adversary should not be able to distinguish between the encryption of two challenge keywords of its choice, even if it is allowed to obtain trapdoors for any non-challenge keywords. Formally, we associate to any adversary A and a bit $b \in \{0, 1\}$ the following experiment:

$$
\begin{array}{l|l}
\textit{Experiment } Exp_{PEKS,A}^{peks-ind-cpa-b} & \\
WSet \leftarrow \phi;\ (pk,\ sk) \leftarrow KeyGen(1^k) & \\
\textit{pick random oracle } H & \textit{Oracle } TRAPD(w) \\
(w_0, w_1, state) \leftarrow A^{TRAPD(\cdot),H}(find, pk) & WSet \leftarrow WSet \cup \{w\} \\
C \leftarrow PEKS^H(pk, w_b) & t_w \leftarrow Td^H(sk, w) \\
b' \leftarrow A^{TRAPD(\cdot),H}(guess, C, state) & \textit{return } t_w \\
\textit{if } \{w_0, w_1\} \cap WSet = \phi \textit{ then return } b' \textit{ else return } 0 &
\end{array}
$$

The PEKS-IND-CPA-*advantage* of A is defined as

$$
Adv_{PEKS,A}^{peks-ind-cpa}(k) = \Pr\left[Exp_{PEKS,A}^{peks-ind-cpa-1}(k) = 1\right] - \Pr\left[Exp_{PEKS,A}^{peks-ind-cpa-0}(k) = 1\right]
$$

A scheme PEKS is said to be PEKS-IND-CPA-*secure* if the above advantage is a negligible function in k for all PTAs (Polynomial Time Adversary) A.

Theorem 1. The PEKS scheme proposed in *Pre-Processing* is PEKS-IND-CPA-secure assuming that the BDH problem is hard relative to generator g.

See literature [12] for the detailed proof.

Any cryptographic primitive must meet two conditions. One is of course a security condition. The other, which we call a consistency condition, ensures that the primitive fulfills its function. In a PEKS scheme, Alice can provide a gateway with a trapdoor t_w (computed as a function of her secret key) for any keyword w of her choice. A sender encrypts a keyword w' under Alice's public key pk to obtain a ciphertext C that is sent to the gateway. The latter can apply a test function $Test$ to t_w, C to get back 0 or 1. The consistency condition is that if $w = w'$ then $Test$ (t_w, C) returns 1 and if $w \neq w'$ it returns 0.

To define consistency, we take an approach like security condition. Namely, we imagine the existence of an adversary U that wants to make consistency fail. More precisely, let $PEKS = (KG, PEKS, Td, Test)$ be a PEKS scheme. We associate to an adversary U the following experiment:

$$
\begin{array}{c}
\textit{Experiment } Exp_{PEKS,U}^{peks-consist}(k) \\
(pk,\ sk) \leftarrow KG(1^k);\ \textit{pick random oracle } H \\
(w,\ w') \leftarrow U^H(pk);\ C \leftarrow PEKS^H(pk,\ w);\ t_{w'} \leftarrow Td^H(sk,\ w') \\
\textit{if } w \neq w' \textit{ and } Test^H(t_{w'},\ C) = 1 \textit{ then return } 1 \textit{ else return } 0
\end{array}
$$

We define the advantage of U as

$$Adv_{PEKS,U}^{peks-consist}(k) = \Pr\left[Exp_{PEKS,U}^{peks-consist}(k) = 1\right]$$

where the probability is taken over all possible coin flips of all the algorithms involved, and over all possible choices of random oracle H. The scheme is said to be perfectly consistent if this advantage is 0 for all (computationally unrestricted) adversaries U, statistically consistent if it is negligible for all (computationally unrestricted) adversaries U, and computationally consistent if it is negligible for all PTAs U.

Theorem 2. The PEKS scheme proposed in *Pre-Processing* is statistically consistent.

See literature [12] for the detailed proof.

Theorem 3. If the signature scheme is existentially unforgeable and the BDH problem is hard, no adversary against the soundness of our public-verification scheme could cause verifier to accept in a proof-of-retrievability protocol instance with non-negligible probability, except by responding with correctly computed values.

Theorem 4. Suppose a cheating prover on a n-block file F is well-behaved in the sense above, and that it is ε-admissible. Let $\omega = 1/\#B + (\rho n)^l/(n - c + 1)^c$. Then, provided that $\varepsilon - \omega$ is positive and non-negligible, it is possible to recover a ρ-fraction of the encoded file blocks in $O(n/(\varepsilon - \rho))$ interactions with cheating prover and in $O(n^2 + (1 + \varepsilon n^2)(n)/(\varepsilon - \omega)$ time overall.

Theorem 5. Given a fraction of the n blocks of an encoded file F, it is possible to recover the entire original file F with all but negligible probability.

Due to space limitations, the detailed proofs of Theorems 3, 4 and 5 are provided in [13].

EFFICIENCY ANALYSIS

Let X, H, A, M, E, and P respectively denote XOR operation, Hash operation, Addition, Multiplication, Exponentiation, and Pairing operation. The amounts of calculation needed in each process of the proposed scheme are listed in Table I.

Through the analysis in Table I, most of calculations have the computation complexity of $O(1)$ except Hash operation, Multiplication, and Exponentiation in the process of signature generation has a $O(n)$, and this process is completed offline, which has no negative influence on online running efficiency. Clearly, the proposed scheme has a high efficiency on Ciphertext-Searching, and the task of proof verification is delegated to TPA, without expending our own computation resources. Therefore, our scheme is feasible in computational efficiency.

CONCLUSIONS AND PERSPECTIVES

We defined and constructed a cryptographic cloud storage scheme with public verifiability, to support moving the data repository of YML framework to a public cloud infrastructure where the service provider is not completely trustworthy. A statistically consistent PEKS scheme was utilized to figure out the issue of

Table 1. The amounts of calculation needed in each process.

Calculation	pre-processing	Signature generation	Ciphertext searching	Proof generation	Proof verification
X	1	0	1	0	0
H	4	n	4	0	c
A	0	0	0	$c-1$	0
M	0	n	0	$2c-1$	c
E	3	$2n$	1	c	$c+1$
P	1	0	1	0	2

Note: n denotes the number of all the data blocks, and c denotes the number of the blocks to be checked.

ciphertext-searching, which is PEKS-IND-CPA-secure based on BDH problem. A soundness verification scheme was proposed, and then the TPA takes on the burden of integrity verification by challenging the cloud server.

The support for data dynamics via the most general forms of data operation, such as block modification, insertion and deletion, is also a significant step toward practicality, since services in Cloud Computing are not limited to archive or backup data only. While prior works on ensuring remote data integrity often lacks the support of dynamic data operations, so we will concentrate on this point in future work. Also, we will improve the proposed storage scheme by deploying and testing it in the simulated cloud environment. The testing phase will bring new perspectives and will show needed adaptation of the current methods.

ACKNOWLEDGEMENTS

This paper is supported by National Natural Science Foundation of China: "Research on Trusted Technologies for The Terminals in The Distributed Network Environment" (Grant No. 60903018) and "Research on the Security Technologies for Cloud Computing Platform" (Grant No. 61272543).

NOTES ON CONTRIBUTORS

Xin Lv is a PhD candidate in the College of Computer and Information at Hohai University. His research is mainly focused on digital signature applications and security in e-commerce.

Feng Xu is a Professor in the College of Computer and Information at Hohai University. His main research interests are trusted computing and network information security.

Serge G. Petiton is a Tenured Professor in University of Sciences and Technology of Lille. His current research areas include high performance computing language and framework, parallel and distributed computing.

REFERENCES

[1] YML Project Page. http://yml.prism.uvsq.fr

[2] Delannoy, O., & Petiton, S. (2004). A Peer to Peer computing framework: Design and performance evaluation of YML. *Proceedings of third international workshop on parallel and distributed computing, 2004* (pp. 362–369). Los Alamitos, CA: IEEE Computer Society.

[3] Delannoy, O., Emad, N., & Petiton, S. (2006). Workflow global computing with YML. *Proceedings of the 7th IEEE/ACM international conference on grid computing*, *GRID'06* (pp. 25–32). New York: ACM Press.

[4] Choy, L., Delannoy, O., Emad, N., & Petiton, S. (2009). Federation and abstraction of heterogeneous global computing platforms with the YML framework. *Proceedings of international conference on complex, intelligent and software intensive systems*, *CISIS'09* (pp. 451–456). Los Alamitos, CA: IEEE Computer Society.

[5] Shang, L., Petiton, S., Emad, N., Yang, X. L., & Wang, Z. H. J. (2009). Extending YML to be a middleware for scientific cloud computing. *Proceedings of first international conference on cloud computing*, *CloudCom 2009* (Vol. 5931, pp. 662–667). Berlin: Springer-Verlag, LNCS.

[6] Shang, L., Petiton, S., Emad, N., & Yang, X. L. (2010). YML-PC: A reference architecture based on workflow for building scientific private clouds. *Cloud computing principles, systems and applications*, Part 2 (Vol. 0, pp. 145–162). Berlin: Springer-Verlag.

[7] Lv, Xin, Petiton, Serge G., Shang, Ling, Wang, Zhijian, & Xu, Feng (2012). Cryptographic methods for securing the YML framework. 1st International conference on systems and computer science, ICSCS 2012. Villeneuve d'Ascq, France, August 29–31.

[8] Shacham, H., & Waters, B. (2008). Compact proofs of retrievability. In J. Pieprzyk (Ed.), *ASIACRYPT 2008* (Vol. 5350, pp. 90–107). Heidelberg: Springer, LNCS.

[9] Boneh, Dan, & Franklin, Matthew K. (2003). Identity based encryption from the Weil pairing. *SIAM Journal on Computing*, *32*(3), 586–615.

[10] Boneh, D., Lynn, B., & Shacham, H. (2001). Short signatures from the Weil pairing. In C. Boyd (Ed.), *ASIACRYPT 2001* (Vol. 2248, pp. 514–532). Heidelberg: Springer, LNCS.

[11] Juels, A., & Kaliski, B. S. Jr (2007). Pors: proofs of retrievability for large files. *Proceeding of CCS 2007* (pp. 584–597). New York: ACM Press.

[12] Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H. (2005). Searchable encryption revisited: Consistency, properties, relation to anonymous IBE, and extensions. *Advances in Cryptology - CRYPTO 2005* (Vol. 3621, pp. 205–222). Berlin: Springer-Verlag, LNCS.

[13] Wang, Q., Wang, C., Li, J., Ren, K., & Lou, W. Enabling public verifiability and data dynamics for storage security in cloud computing. *Cryptology ePrint Archive*. https://eprint.iacr.org/2009/281.pdf