# A robust cloud access scheme with mutual authentication

Chin-Ling Chen[1,2,a], Yong-Yuan Deng[3], Kun-hao Wang[2] and Chun-Long Fan[4]

[1]*Department of Computer Science and Information Engineering, Chaoyang University of Technology, Taiwan*
[2]*School of Information Engineering, Changchun University of Science And Technology, China*
[3]*Department of Information Management, Chaoyang University of Technology, Taiwan*
[4]*School of Computer Science,Shenyang Aerospace University, China*

**Abstract.** Due to the progress of network technology, we can access some information through remote servers, and we also can save and access lots of personal data in remote servers. Therefore, to protect these data and resist un-authorized access is an important issue. Some researchers proposed authentication scheme, but there still exist some security weaknesses. This article is based on the concept of HDFS (Hadoop Distributed File System), and offers a robust authentication scheme. The proposed scheme achieves mutual authentication, prevents re-play attack, solves asynchronous issue, and prevents offline password guessing attack.

## 1 Introduction

### 1.1 Background

Due to the rapid development of hardware net-work environment, a variety of services and applications began to attract personal attention in today's social environment. In earlier, because of the smaller scale of services, fewer numbers of users, a single server is sufficient to provide all the necessary services.

With the rise of integrated services, single company can provide diverse services, and even operate as cross-industry alliance type. User can use single account to access various services. On the other hand, the huge amount of users is impossible to use single server to satisfy various services. Therefore, we need multi-server to handle these services, and that's a concept of cloud. [7, 8, 11]

### 1.2 Cloud architecture

There are three kinds of architectures about user and server in cloud architecture [9].

#### *1.2.1 All users connect to a single server for authentication and communication*

This service model is simple and easy to set up. But the number of users provided service con-tents increasingly, it will be no longer a single server can handle it, and such architecture is vulnerable to be braked by malicious people.

#### *1.2.2 All users through the gateway and lead to a different server*

Such service model can effectively solve the excessive burden of a single server problem, but this will significantly reduce the reliability of the system architecture, and it is also vulnerable to be braked by malicious people.

#### *1.2.3 All users process identity verification with front server and backend server*

Such service mode can also effectively solve the excessive burden of a single server problem, and the user's real identity is stored in the backend server. It's hard to be braked by malicious peo-ple, the architecture is similar to HDFS (Hadoop Distributed File System). [6, 10]

## 2 Literature review

### 2.1 Review of Nivethitha et al.'s scheme [5]

#### *2.1.1 Registration stage of Nivethitha et al.'s scheme*

Step 1: User selects his/her identity $ID$, password $PW$, and then sends $(ID, PW)$ to register server through secure channel. Registration server forwards the same message to backend server.

Step 2: Register server generates one time password $OTP$, and use the symmetric key $OTP$ to encrypt the

[a]Corresponding author: clc@cyut.edu.tw

$PW$ , $C1 = E_{OTP}(PW)$ , $C2 = C1 \bmod 26$ , and stores $C2$ in its database securely, then sends $OTP$ to backend server through secure channel.

Step 3: Backend server calculates $C3 = E_{PW}(OTP)$, and then stores $(ID, C3)$ in its database securely.

#### *2.1.2 Authentication stage of Nivethitha et al.'s scheme*

Step 1: User inputs his/her identity $ID'$, then sends it to register server. Register server forwards the same message $ID'$ to backend server.

Step 2: Backend server checks its database to get $(ID, C3)$, which stored during registration stage, and then sends $C3$ to register server. Register server forwards the same message $C3$ to user.

Step 3: User calculates $OTP' = D_{PW}(C3)$, and then sends $OTP'$ to register server.

Step 4: Register server calculates $C1 = C2 \bmod 26$, and use the decryption key $OTP$ to decrypt $PW$, $PW = D_{OTP'}(C1)$. Afterward, uses the $PW$ to encrypt $OTP'$, $C4 = E_{PW}(OTP')$, and then sends $C4$ to backend server.

Step 5: Backend server checks $C4 \stackrel{?}{=} C3$. If pass the verification, the backend server gets $ID$ which stored in registration stage, and then sends $ID$ to register server.

Step 6: Register server checks $ID \stackrel{?}{=} ID'$. If pass, it generates $OTP^{new}$ and calculates $C1^{new} = E_{OTP^{new}}(PW)$ , $C2^{new} = C1^{new} \bmod 26$ , and then updates $C2$ into $C2^{new}$ in its database securely for next access.

Later, Mrudula et al. [4] pointed out that Nivethitha et al.'s scheme can't prevent offline password guessing attack. When an attacker intercepts the messages $ID'$, $C3$ and $OTP'$ which transferred between user and register server during authentication stage, he/she can perform dictionary attack to guess a password $PW'$ , and calculate $OTP'' = D_{PW'}(C3)$. If $OTP''$ is equal to $OTP'$, it means he/she gets the correct password successfully, and can be a legal user for future access.

### 2.2 Review of Mrudula et al.'s scheme

#### *2.2.1 Registration stage of Mrudula et al.'s scheme*

Step 1: User selects his/her identity $ID$, password $PW$ and a random number $a$, uses a hashing function $h(\ )$ and exclusive OR operation $\oplus$ to calculate $APW = h(a \oplus PW)$, and then sends $(ID, APW)$ to register server through secure channel.

Step 2: Register server generates $OTP$, and calculates $CID = ID \oplus h(OTP)$ , $C1 = APW \oplus h(OTP)$ , $C2 = C1 \bmod 26$ , and then sends $(CID, APW, OTP)$ to backend server through secure channel.

Step 3: Backend server calculates $C3 = h(APW) \oplus OTP$ , $ID = CID \oplus h(OTP)$, and then stores $(ID, C3)$ in its database securely.

Step 4: Register server generates a random number $ru$, calculates $HID = h(ID||X)$ and $R$, $R = ru \oplus h(X||ID)$ , stores $(HID, C2, R)$ in its database securely, and then sends $ru$ to user through secure channel.

#### *2.2.2 Authentication stage of Mrudula et al.'s scheme*

Step 1: User inputs his/her identity $ID'$, then sends it to register server. Register server forwards the same message $ID'$ to backend server.

Step 2: Backend server checks its database to get $(ID, C3)$ which stored during registration stage, and then sends $C3$ to register server. Register server forwards the same message $C3$ to user.

Step 3: User calculates $OTP' = C3 \oplus h(APW)$ , $C4 = OTP' \oplus h(ru||ID')$ , and then sends $C4$ to register server.

Step 4: Register server calculates $ru = R \oplus h(X||ID)$ , $OTP' = C4 \oplus h(ru||ID')$ , $C1 = C2 \bmod 26$ , $APW = C1 \oplus h(OTP')$, $C5 = h(HPW) \oplus OTP'$, and then sends $C5$ to backend server.

Step 5: Backend server checks $C5 \stackrel{?}{=} C3$. If pass, backend server gets $ID$ from $(ID, C3)$ , which stored during registration stage, and then sends $ID$ to register server.

Step 6: Register server checks $ID \stackrel{?}{=} ID'$ . If pass, it generates $ru^{new}$ , updates $ru$ into $ru^{new}$ in its database securely for next access, and calculates $C6 = h(APW) \oplus ru^{new}$, then sends $C6$ to user.

Mrudula et al.'s scheme solves offline password guessing attack successfully due to they use a random number $ru$ between user and register server during authentication stage. The random number $ru$ changes after a success round, so an attacker can't perform offline password guessing attack simply. We find that their scheme exists another problem that is asynchronous issue.

When an attacker intercepts the final message between user and register server during authentication stage, they can't communicate anymore in the future. User and register server hold different random number $ru$ , and which is decided from register server, there is no way to let user gets another random number again. Besides, both Nivethitha et al. and Mrudula et al.'s scheme doesn't achieve mutual authentication. [1, 2, 3]

## 3 The proposed scheme

### 3.1 Registration stage

Step 1: User selects his/her identity $ID$, password $PW$ and a random number $ru$, calculates

$$HPW = h(PW) \quad (1)$$

and then sends $(ID, HPW, ru)$ to register server through secure channel.

Step 2: After receiving the message, register server generates $OTP$ and calculates

$$EID = ID \oplus h(OTP) \quad (2)$$

$$C1 = HPW \oplus h(OTP) \quad (3)$$

$$C2 = C1 \bmod 26 \quad (4)$$

and then sends $(EID, HPW, OTP)$ to backend server through secure channel.

Step 3: Backend server calculates

$$C3 = h(HPW) \oplus h(OTP) \quad (5)$$

$$ID = EID \oplus h(OTP) \quad (6)$$

and then stores $(ID, C3)$ in its database securely.

Step 4: Register server generates a secret parameter $x$, calculates

$$HID = h(ID \oplus x) \quad (7)$$

$$C4 = ru \oplus h(ID) \oplus h(x) \quad (8)$$

and stores $(HID, C2, C4)$ in its database securely, then sends $HID$ to user through secure channel.

Step 5: User stores $(HID, ru)$ in his/her device securely.

### 3.2 Authentication stage

Step 1: User inputs his/her identity $ID'$, and then sends it to register server. Register server forwards the same message $ID'$ to backend server.

Step 2: Backend server checks its database to get $(ID, C3)$ which stored during registration stage, and then sends $C3$ to register server. Register server forwards the same message $C3$ to user.

Step 3: After receiving the message, user calculates

$$h(OTP') = C3 \oplus h(HPW) \quad (9)$$

$$C5 = h(OTP') \oplus h(ID' \oplus ru) \quad (10)$$

generates a new random number $ru^{new}$, calculates

$$C6 = ru \oplus ru^{new} \quad (11)$$

and then sends $(C5, C6, HID)$ to register server.

Step 4: Register server checks

$$HID \overset{?}{=} h(ID' \oplus x) \quad (12)$$ Note [a]

If pass, the register server calculates

$$ru = C4 \oplus h(ID) \oplus h(x) \quad (13)$$

$$h(OTP') = C5 \oplus h(ID' \oplus ru) \quad (14)$$

$$C1 = C2 \bmod 26 \quad (15)$$

$$HPW = C1 \oplus h(OTP') \quad (16)$$

$$C7 = h(HPW) \oplus h(OTP') \quad (17)$$

and then sends $C7$ to backend server.

Step 5: Backend server checks

$$C7 \overset{?}{=} C3 \quad (18)$$

If pass, backend server gets $ID$ which stored during registration stage. Backend server calculates

$$C8 = C3 \oplus h(ID) \quad (19)$$

and then sends $C8$ to register server.

Step 6: Register server checks

$$C8 \overset{?}{=} C7 \oplus h(ID') \quad (20)$$

If pass, register server calculates

$$ru^{new} = C6 \oplus ru \quad (21)$$

$$C9 = h(HPW) \oplus ru^{new} \quad (22)$$

$$C4^{new} = ru^{new} \oplus h(ID) \oplus h(x) \quad (23)$$

and then sends $C9$ to user.

Step 7: User checks

$$ru^{new} \overset{?}{=} C9 \oplus h(HPW) \quad (24)$$

If pass, user updates $ru$ into $ru^{new}$ for next access.

## 4 Security analysis

### 4.1 Achieve mutual authentication

During the login phase, the user, register server and backend server authenticate each other to make sure the legality of each roles. Register server uses $HID \overset{?}{=} h(ID' \oplus x)$ to authenticate user, backend server uses $C7 \overset{?}{=} C3$ to authenticate register server and user, register server uses $C8 \overset{?}{=} C7 \oplus h(ID')$ to authenticate backend server, and user uses $ru^{new} \overset{?}{=} C9 \oplus h(HPW)$ to authenticate register server and backend server. The proposed scheme achieves mutual authentication.

### 4.2 Prevent replay attack

During login phase, an attacker may intercept the message $C5 = h(OTP') \oplus h(ID' \oplus ru)$, $C6 = ru \oplus ru^{new}$, $HID = h(ID \oplus x)$ that sent to register server, and sends it again in the future. The attacker can't success because of $C5 = h(OTP') \oplus h(ID' \oplus ru)$ includes a random value $ru$ is changed in every round.

When the attacker sends $C5 = h(OTP') \oplus h(ID' \oplus ru)$ to register server again, register server computes $h(OTP') = C5 \oplus h(ID' \oplus ru)$ but can't get the original $h(OTP')$ because of the two $h(ID' \oplus ru)$ are different, and can't be recovered by exclusive-or operation. Therefore, the proposed scheme can prevent replay attack.

[a] $A \overset{?}{=} B$ : determine if A equal to B

### 4.3 Solve asynchronous issue

To prevent replay attack, user and register server update a new random value after a successful authentication for next round. When an attacker intercepts the final message between user and register server, it will cause asynchronous problem. Because of user and register server hold different random value, they can't communicate anymore.

In the previous scheme, the random value decided by register server, the asynchronous problem will happen, and there's no way to solve this situation. In our proposed scheme, even the final message $C9$ has been intercepted by an attacker, user can run the registration phase to send $(ID, HPW, ru)$ again to solve the asynchronous issue.

### 4.4 Prevent offline password guessing attack

When an attacker intercepts the messages $ID'$, $C3 = h(HPW) \oplus h(OTP)$, $C5 = h(OTP') \oplus h(ID' \oplus ru)$, $C6 = ru \oplus ru^{new}$, $HID = h(ID \oplus x)$ and $C9 = h(HPW) \oplus ru^{new}$ between user and register server, he/she still can't do anything to get the password. The proposed scheme uses hash function and exclusive-or calculation to protect all the messages. It is computationally impossible to get the password $HPW$, the proposed scheme can prevent offline password guessing attack.

## 5 Security and complexity comparison

The following table 1 and table 2 are security comparison and complexity comparison respectively. Although the proposed scheme has higher complexity of calculation, it's not a problem for today's hardware and network environment. The proposed scheme achieves more security requirements than the previous schemes, especially mutual authentication.

### 5.1 Security comparison table

**Table 1.** Security comparison

| | Nivethitha et al.'s scheme | Mrudula et al.'s scheme | The proposed scheme |
|---|---|---|---|
| Achieve mutual authentication | No | No | Yes |
| Prevent replay attack | Yes | Yes | Yes |
| Solve asynchronous issue | Yes | No | Yes |
| Prevent offline password guessing attack | No | Yes | Yes |

### 5.2 Complexity comparison table

**Table 2.** Complexity comparison

| | Nivethitha et al.'s scheme | Mrudula et al.'s scheme | The proposed scheme |
|---|---|---|---|
| Registration stage (User) | N/A | 1$T_{XOR}$ +1$T_{HASH}$ | 1$T_{HASH}$ |
| Registration stage (Registration server) | 1$T_{ENC}$ +1$T_{MOD}$ | 3$T_{XOR}$ +4$T_{HASH}$ +1$T_{MOD}$ | 5$T_{XOR}$ +5$T_{HASH}$ +1$T_{MOD}$ |
| Registration stage (Backend server) | 1$T_{ENC}$ | 2$T_{XOR}$ +2$T_{HASH}$ | 2$T_{XOR}$ +3$T_{HASH}$ |
| Authentication stage (User) | 1$T_{ENC}$ | 2$T_{XOR}$ +2$T_{HASH}$ | 4$T_{XOR}$ +5$T_{HASH}$ |
| Authentication stage (Registration server) | 2$T_{ENC}$ +1$T_{MOD}$ | 5$T_{XOR}$ +5$T_{HASH}$ +1$T_{MOD}$ | 12$T_{XOR}$ +10$T_{HASH}$ +1$T_{MOD}$ |
| Authentication stage (Backend server) | N/A | N/A | 1$T_{XOR}$ +1$T_{HASH}$ |

$T_{XOR}$: time complexity of exclusive-or operation
$T_{HASH}$: time complexity of hash operation
$T_{MOD}$: time complexity of mod operation
$T_{ENC}$: time complexity of symmetric encryption/decryption

## 6 Conclusions

In the cloud environment, lots of users' sensitive data are stored in the remote servers. Besides legal users, malicious people also want to get these data. There must have a robust authentication protocol to protect these data. Some previous researchers proposed the authentication protocol for HDFS cloud architecture, but they still get some security defects. The proposed protocol solves these defects successfully, and also satisfies more security requirements.

## Acknowledgement

This research was supported by the Ministry of Science and Technology, Taiwan, R.O.C., under contract number

MOST 105-2221-E-324-007, MOST 105-2622-E-212-008 -CC2 and MOST103-2632-E-324-001-MY3.

## References

1. K. Hamlen, M. Kantarcioglu, L. Khan, & ham B. Thuraising. Security issues for cloud computing. *International Journal of Information Security and Privacy.* Apr–Jun; **4(2)**:39-51 (2010)
2. K. Hwang, S. Kulkarni, & Y. Hu. Cloud security with virtualized defense and reputation - based trust management. *Eighth IEEE International Conference on Pervasive Intelligence and Computing, (PICom2009).* Chengdu, China. 717-722. (2009)
3. G. Kousiouris, G. Vafiadis, & T. Varvarigou. A frontend, Hadoopbased data management service for efficient federated clouds. *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*. 511-516. (2011).
4. S. Mrudula, M.R.M. Chandra, S.V. Chandra. A Secure and Light Weight Authentication Service in Hadoop using One Time Pad, *2nd International Symposium on Big Data and Cloud Computing (ISBCC' 15)*, 81-86 (2015).
5. S. Nivethitha, A. Gangaa, & S. Shankar. Authentication Service in Hadoop Using one Time Pad, *Indian Journal of Science & Technology*, **7**, 56-62 (2014).
6. O.M. Owen. Integrating Kerberos into Apache Hadoop Kerberos. *Conference 2010*, 26–27 Oct, MIT, USA (2010).
7. P.K. Rahul, & K.T. Gireesh. A Novel Au thentication Framework for Hadoop, *International conference on Intelligence and Evolutionary Algorithms in Engineering Systems (ICAEES 2014)* , **324**, 333-340. (2015).
8. G.S. Sadasivam, K.A. Kumari, & S. Rubika. A Novel Authentication Service for Hadoop in Cloud Environment, *IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, 1-6, India (2012)
9. S.G. Sudha, K.K. Anitha, & S. Rubika. A novel authentication service for hadoop in cloud environment. *IEEE International Conference on Cloud Computing in Emerging Markets (CCEM).* 16 (2012).
10. G. Turkington. *Hadoop Beginner's Guide.* PACKT Publishing (2013).
11. Q. Zhou, X. Deqin, T. Hunming, & R. Chunming. TSHC: Trusted Scheme for Hadoop Cluster, *Fourth International Conference on Emerging Intelligent Data and Web Technologies (EIDWT)*, 9-11 (2013).