*IJAET International Journal of Application of Engineering and Technology, ISSN: 2395-3594*

*Vol-1 No.-2*

# KECCAK ATTRIBUTE BASED ENCRYPTION IN CLOUD

Ashish Yadav[a], Deepak Choudhary[a], and PawanTiwari3[a]

1[a]M.Tech Scholar 1, IET College, Alwar and India
2[a]Assistant Professor 2, IET College, Alwar and India
3[a] M.Tech Scholar 1, IET College, Alwar and India

ashish7066@gmail.com, deepakchaudhary1707@gmail.com, pawantiwaari@gmail.com

**ABSTRACT**

**The advancement in the field of cloud computing made more and more enterprises to join the cloud computing putting aside the traditional methods for sharing sensitive data by outsourcing it. To maintain the confidentiality against the untrusted source and service provider, the best is to store the data by encrypting. The obstacle in the way is how to provide the access to the employee at various levels within the cloud (access policies and rights). Paper aim to solve the problem by first propose a hierarchical attribute – based encryption scheme and cipher text policy attribute based encryption (CP-ABE) system, to provide control.**

***Keywords:*** *- Attribute, Keccak, SHA-3, KDC (Key Distribution Authority)*

## I. INTRODUCTION

ABE was proposed by Sahai and Waters. In ABE, a user has a set of attributes in addition to its unique ID. There are two classes of ABEs. In Key-policy ABE or KP-ABE (Goyal*et al.*), the sender has an access policy to encrypt data. A writer whose attributes and keys have been revoked cannot write back stale information. The receiver receives attributes and secret keys from the attribute authority and is able to decrypt information if it has matching attributes. In Cipher text-policy, CP-ABE, the receiver has the access policy in the form of a tree, with attributes as leaves and monotonic access structure.

All the approaches take a centralized approach and allow only one KDC, which is a single point of failure. Chase proposed a multi-authority ABE, in which there are several KDC authorities (coordinated by a trusted authority) which distribute attributes and secret keys to users. Multi-authority ABE protocol was studied in which required no trusted authority which requires every user to have attributes from at all the KDCs. Recently, Lewko and Waters proposed a fully decentralized ABE where users could have zero or more attributes from each authority and did not require a trusted server. In all these cases, decryption at user's end is computation intensive. So, this technique might be inefficient when users access using their mobile devices. To get over this problem, Green *et*al.proposed to outsource the decryption task to a proxy server, so that the user can compute with minimum resources (for example, hand held devices). However, the presence of one proxy and one key distributionCenter makes it less robust than decentralized approaches. Both these approaches had who want to remain anonymous while accessing the cloud. To ensure anonymous user authentication Attribute Based Signatures were introduced by Maji*et al*. This was also a centralized approach. A recent scheme by the same authors takes a decentralized approach and provides authentication without disclosing the identity of the users. However, as mentioned earlier in the previous section it is prone to replay attack.

## II. PROPOSED WORK

The main contributions of this paper are the following:

1. Distributed access control in such a way that the only authorized user can have access to the source of data.
2. Confidentiality in regards of the Identity of the user.
3. The architecture is decentralized.
4. The access control and authentication are both collusion resistant, meaning that no two users can collude and access data or authenticate themselves, if they are individually not authorized.
5. Implementation of SHA-3 to overcome drawbacks of SHA1 & SHA-3
6. The proposed scheme is resilient to replay attacks. A writer whose attributes and keys have been revoked cannot write back stale information.
7. The protocol supports multiple read and write on the data stored in the cloud.

On presenting her id the trustee gives her a token. A creator on presenting the token to one or more KDCs

ISSN 2395-3594

receives keys for encryption/decryption and signing. SKs are secret keys given for decryption, Kx are keys for signing. The message MSG is encrypted under the access policy X. The access policy decides who can access the data stored in the cloud. The creator decides on a claim policy Y, to prove her authenticity and signs the messageunder this claim. The cipher text C with signature is c, and is sent to the cloud. The cloud verifies the signature and stores the cipher text C. When a reader wants to read, the cloud sends C. If the user has attributes matching with access policy, it can decrypt and get back original message. Write proceeds in the same way as file creation. By designating the verification process to the cloud, it relieves the individual users from time consuming verifications. When a reader wants to read some data stored in the cloud, it tries to decrypt it using the secret keys it receives from the KDCs. If it has enough attributes matching with the access policy, then it decrypts the information stored in the cloud.

*A. Creation of KDC*

Different number of KDC's are created and to register a user details. KDC name, KDC id and KDC password are given as input to create KDC. Inputs will save in a database and to register a user details given a input as username and user id.

*B. KDC Authentication*

After KDC give a user id to a user, the user will enroll the personal details to KDC's given an input as user name, user id, password etc. The KDC will verify the user details and it will save it in Database.

*C. User Accessibility*

Users can get the token from trustee for the file upload. After trustee was issuing a token, trustee can view the logs. User can login with their credentials and request the token from trustee for the file upload using the user id. After the user id received by the trustee, trustee will be create token using user id, key and user signature (SHA).

*D. Generation of access policy*

After the key was received by the User, the message MSG is encrypted under the access policies. The access policies decide who can access the data stored in the cloud. The cipher text C with signature c, and is sent to the cloud. The cloud verifies the signature and stores the cipher text C. When a reader wants to read, the cloud sends C. If the user has attributes matching with access policy, it can decrypt and get back original message and user can upload the file after user get key from the KDC.

*E. File accessing*

Using their access policies the users can download their files by the help of kdc's to issue the private keys for the particular users. After trustee token issuance for the users, the users produce the token to the KDC then the token verify by the KDC if it is valid then KDC will provide the public and Private key to the user. After users received the keys the files are encrypt with the public keys and set their Access policies (privileges).

*F. File Restoration*

Files stored in cloud can be corrupted. So for this issue, using the file recovery technique to recover the corrupted file successfully and to hide the access policy and the user attributes.

*G. Secure Hash Algorithm*

Definition: SHA-3 after a research for near a decade the result has provided with highly significant Hash Algorithm that is a advancement over the previous scheme that is SHA-1which was used to find out that weather the file or information is altered or not. An SHA-2 cryptographic hash function is also available.

One iteration within the SHA-1 compression function. A, B, C, D and E are 32bit words of the state. *F* is a nonlinear function that varies. *n*denotes a left bit rotation by *n* places. *n*varies for each operation. Wt is the expanded message word of round t. Kt is the round constant of round t. denotes addition modulo 232.

*H. Paillier Algorithm*

The Paillier cryptosystem, named after and invented by Pascal Paillier is a probabilistic asymmetric algorithm for public key cryptography.

*Key generation* Choose two large prime number *p* and *q* randomly and independently of each other such that gcd (pq,(p-1)(q-1))=1. This property is assured if both primes are of equivalent length, i.ep,q {0,1 }s-1 for security parameter S . Compute n=pq and λ=lcm(p-1,q-1).

Select random integer g where gZ*n2. Ensure n divides the order of g by checking the existence of the following modular multiplicative inverse .μ= (L(gλ mod n2))-1mod n, where function L is defind as The public (encryption) key is (n,g).. The private (decryption) key is (λ,μ ).

*Encryption* Let m be a message to be encrypted where m Zn. Select random r where r Z*n. Compute cipher text as: c= gm .rn mod n2 *Decryption* Cipher text: cZ*n2.Compute message: m =L(cλ mod n2). M mod n

ISSN 2395-3594

*Decryption* Cipher text: $cZ^*_{n2}$.Compute message: m =$L(c^{\lambda} \bmod n_2)$. M mod n

computing putting aside the traditional methods of sharing sensitive data by outsourcing it. To maintain the confidentiality against the untrusted source and service provider, the best is to store the data by encrypting. The obstacle in the way is how to provide the access to the employee at various levels within the cloud(access policies and rights)? Paper aims to solve the problem by first propose a hierarchical attribute – based encryption scheme and cipher text policy attribute based encryption (CP-ABE) system, to provide control.

## III. IMPLANTATION OF SHA-3

The basis of keccak is the 'state' which is a 5 x 5 array of words. A word is a 64-bit value. So, the representation would be 5 x 5 x 64 cube with each small section of the code to be bit. And the steps involved in the implementation of SHA 3 (Keccak Algorithm) is mentioned with the figure:-

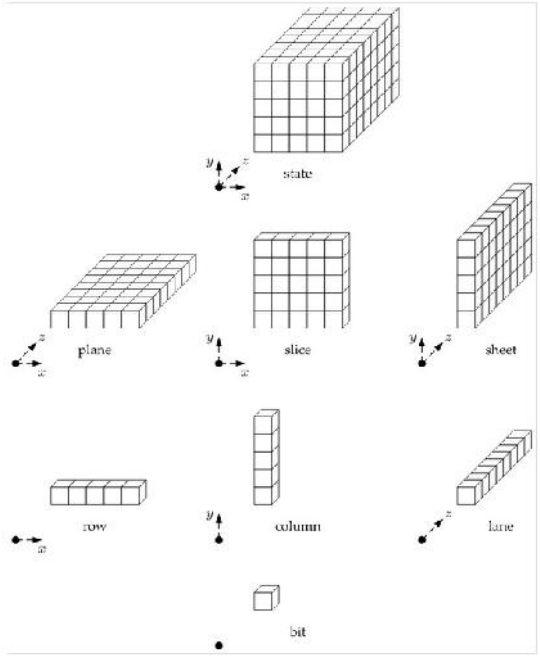


Fig 1: Functions in Keccak

We have five functions to make the above part operational and keccak code in python is mentioned sequentially below:

**1. θ *(Theta Function)***

```
#Theta step
    forx inrange(5):
     C[x] =A[x][0]^A[x][1]^A[x][2]^A[x][3]^A[x][4]
    forx inrange(5):
     D[x] =C[(x-1)%5]^self.rot(C[(x+1)%5],1)
    forx inrange(5):
      fory inrange(5):
        A[x][y] =A[x][y]^D[x]
```

In the image below we have taken 'sum' of one column and put it with the 'sum' of other column and then 'adding ' that into a bit. Well any addition in this is actually xor. Now the first part of the code example takes each sheet (which is an array size 5) and xor all of them together, creates an 'xor' lane that is put that into an array of size 5(called C). Now the next part of the code is to create a 'dual xor' lane. Now lets say you want to modify A[0][x]. You can't C[4] ^ C[1] since you need the 2nd bit of C[4] xored with the 1st bit of C[1]. So a modified version of C[1] is created that is shifted one bit to the left. The now modified 2nd bit of C[1] is the 1st bit of the original C[1]. These 'dual xor' lanes are put into D(an array size 5) Then the last step in the code that that D[x] is xored with every element such that A[x][0..4].

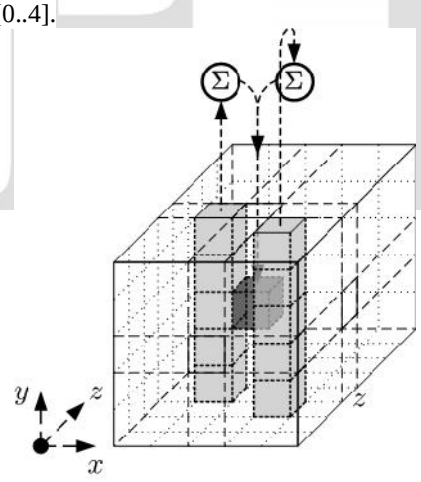


Fig 2: XOR operation between columns

***2.ρ*(Rho)**

*Note: Rho and Pi are done together in the python code but separated here.*

```
## Rotation offsets
  r=[[0,  36,  3,  41,  18]  ,
    [1,  44,  10,  45,  2]  ,
    [62,  6,  43,  15,  61]  ,
    [28,  55,  25,  21,  56]  ,
    [27,  20,  39,  8,  14]  ]

#Rho step
```

ISSN 2395-3594

```
forx inrange(5):
    fory inrange(5):
     B[x][y] =self.rot(A[x][y], self.r[x][y])
```

n this step there is a set array called 'r' that rotates each lane in the state so that if the look at the left-most, top-most lane(which rotates one bit) in the picture the 1st bit becomes the 2nd bit and the last bit has become the first bit.

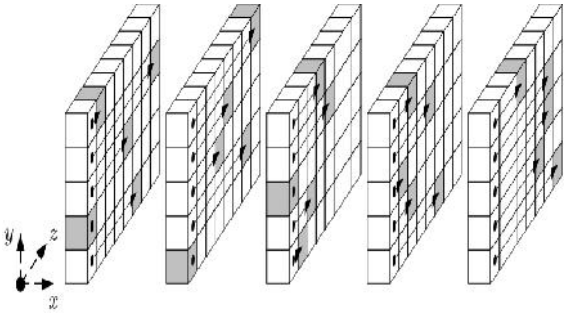

Fig 3: Shifting of Bits by 1to Right.

***3.π*(Pi)**

```
#Pi step
forx inrange(5):
  fory inrange(5):
    B[y][(2*x+3*y)%5] =B[x][y]
```

It took a while for me to understand this code with the provided picture. The reason was that I was thinking that x = y = 0 was in the bottom-left corner. It's not. It's the center element. So the element below the center element is B[0][4] and the one to the left of the center is B[4][0]. The math for this function is fairly easy to understand when you look at the pseudo-code

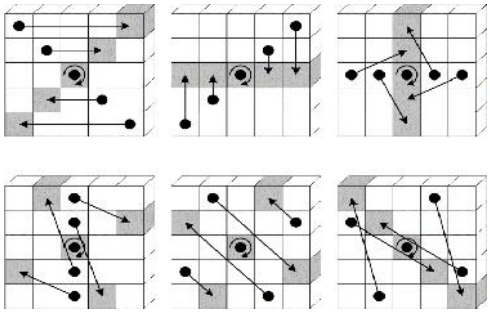

Fig 4: Pseudo –Code Processing

***4.χ*(Chi)**

```
#Chi step
forx inrange(5):
fory inrange(5):
A[x][y] =B[x][y]^((~B[(x+1)%5][y])
& B[(x+2)%5][y])
```

This operation also a simple operations of bit-and/not/xor. Here is a representation. If A(B[0][0]) is the word you are trying to transform into Å(post Chi) then Y(B[1][0]) is the next element to A's left and Z(B[2][0]) then the equation is (A xor ((not Y) and Z)) = Å

The last step does not have a image to go with it but is based on a fixed 'Round Constant' matrix and mixed with certain values like the current round number and the size of the lanes

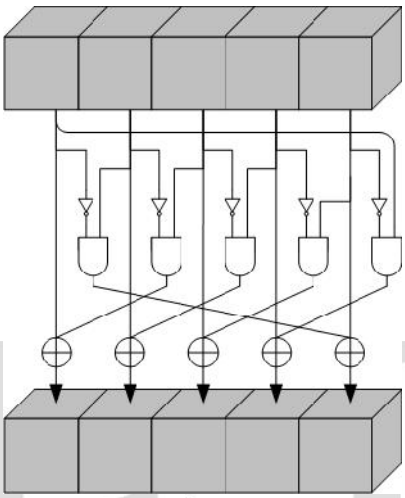

Fig 5: Operations of bit-AND/NOT/XOR

```
5.#Iota step

#i = round number
    #Compute lane length (in bits)
      w=(r+c)//25
      RCfixed =self.RC[i]%(1<<w)
         B[0][0] =B[0][0]^RCfixed
```

The you can return B which is the modified A for one round. Those are the basic round steps for SHA-3. Hopefully this helps someone understand a little more

## IV. ANALYSIS

A notable problem with SHA-1 and SHA-2 is that they both use the same engine, called Merkle-Damgard, to process message text. This meansthat a successful attack on SHA-1 becomes a potential threat on SHA-2.

Consider SHA-1 for instance. A brute force attack usually takes at least $2^{80}$ rounds (a round is a single cycle of transformation of the interim hash value) to find a collision in a full-round SHA-1. But in February 2005, Xiaoyun Wang and colleagues used a differential path attack to break a full-round SHA-1, and it took only $2^{69}$ cycles to succeed. That same attack was later corroborated by Martin Cochran in August 2008.

ISSN 2395-3594

In 2012, Mark Stevens used a series of cloud servers to perform a differential path attack on SHA-1. His attack produced a near-collision after $2^{58.5}$ cycles. He also estimated a modified attack can manage a full-collision after $2^{61}$ cycles.
As to SHA-2, the only successful attacks were those against a limited round SHA-2 hash. The most effective attack was against a 46-round SHA-2 (512-bit variant) and against a 41-round SHA-2 (256-bit variant). It took $2^{253.6}$ cycles to break the 256-bit variant and $2^{511.5}$ cycles for the 512-bit variant.
The fact remains that, while no successful attacks against a full-round SHA-2 have been announced, there is no doubt that attack mechanisms are being developed in private. This is one reason why NIST sponsored the SHA-3 competition, which led to the development and recent adoption of Keccak.

## V. CONCLUSION

The concept behind the paper is to reduce the chances collision among the hash key and decentralized the Key Distributing Authority (KDC).
The SHA-3 algorithm is a result of long research over the drawbacks of earlier version of Hash function (SHA-0, SHA-1 & SHA-2).

## VI. REFERENCE

[1]. Minu George, International Journal of "Advanced Research in Computer and Communication Engineering" on "A Survey on Attribute Based Encryption Scheme in Cloud Computing"2013.

[2]. Cheng-Chi Lee, "International Journal of Network Security", on "A Survey on Attribute-based Encryption Schemes of Access Control in Cloud Environment", Vol.15, No.4, PP.231-240, July 2013.

[3]. Guojun Wang, "Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers" in Elsevier Journal, 2011.

[4]. Junzuo lai, "Information forensics and security, ieee transactions" on "attribute based encryption with verifiable out sourced decryption" (volume:8, issue: 8 ),2013.

[5]. Bethencourt, j. "Security and privacy, 2007. sp '07. ieee symposium on" on ciphertext-policy attribute-based encryption,2007.

## Biographies

**Ashish Yadav**
**M.Tech Scholar**
**Computer Science Engineering**
**IET College Alwar (Raj.)**

**Deepak Choudhary**
**Assistant Professor**
**Computer Science Engineering**
**IET College Alwar (Raj.)**

**Pawan Tiwari**
**M.Tech Scholar**
**Computer Science Engineering**
**IET College Alwar (Raj.)**

ISSN 2395-3594