IET Information Security

Research Article

IET Journals
The Institution of Engineering and Technology

Dynamic authenticated data structures with access control for outsourcing data stream

ISSN 1751-8709
Received on 15th June 2015
Accepted on 4th October 2016
E-First on 30th November 2016
doi: 10.1049/iet-ifs.2015.0243
www.ietdl.org

Yi Sun[1,2,3] ✉, Xingyuan Chen[2], Xuehui Du[2,3], Jian Xu[2]

[1]*Beijing Jiaotong University School of Computer & Information Technology, Beijing 100044, People's Republic of China*
[2]*Zhengzhou Information Science and Technology Institute, Zhengzhou 450004, People's Republic of China*
[3]*State Key Laboratory of Mathematics Engineering and Advanced Computing, Zhengzhou 450004, People's Republic of China*
✉ *E-mail: 11112072@bjtu.edu.cn*

Abstract: Today, many resource-constrained terminal devices prefer to outsource data stream to an intermediary for managing and storing. However, within this growing trend, the trusted problem of outsourcing data stream is universally concerned. It is extremely critical to prove that the data stream provided by the third party is trust. Therefore, in order to efficiently and effectively verify the trusted of the outsourcing data stream with adequate control, this study presents a kind of dynamic authenticated data structure with access control on outsourced data stream. Based on this data structure, the authors are able to establish a novel authentication scheme, which can support data stream to add and update in real time and verification with fine-grained access control. In addition, the security and efficiency of the proposed scheme are analysed in this study. Through comparing and analysing with the existing schemes, the proposed scheme has higher security and efficiency in terms of data stream addition and update.

1 Introduction

Many applications produce large amounts of data streams, such as Web access analysis, sensor network monitoring, real-time monitoring, distributed intrusion detection and stock trading. Considering the cost and expense, when these data stream are generated, they are often outsourced to an intermediary (such as cloud, outsourcing service providers) for managing and storing. However, the intermediary may expose to a public and untrusted environment that has potential leakage of information disclosure, tampering and unauthorised access. With all these security weaknesses, the trust and integrity verification of outsourcing data stream is required. A simple method to implement is that clients downloaded the data stream from intermediary and verified them. However, data stream is set of data sequence with dynamic, continuous, high arriving speed features, this method is unfeasible. In order to achieve efficient and secure verification on outsourced data stream poses several challenges:

- With real-time authentication, it is unable to store the entire data stream before verifying the data stream.
- Consider dynamic variability requirements, verification of outsourcing data stream should support dynamic updating and real-time increasing of outsourcing data stream.
- In many applications, we not only to verify the integrity of the data, but also to verify the order of data stream. For example, ID number, bank account number, and DNA. It would be a disaster if order of these numbers is misplaced.
- Verification of outsourcing data stream should be controlled. In many applications, either same or different domains, not all users can access and verify outsourced data stream. But the existing schemes are either public or private verification, and there are some risks that security key exposure or over scale verification.

To solve the above problems, we proposed a novelty dynamic authenticated data structure with access control, which ensure that the verification of outsourcing data stream is more effective and secure.

1.1 Our contributions

The main contributions of our work are summarised as follows:

- We proposed a novelty dynamic authenticated data structure with access control on outsourced data stream. This data structure is essentially an authentication tree composed of an arbitrary hash function, multiple-collision trapdoor hash function and Ciphertext-policy attribute-based encryption (CP-ABE). Based on the authenticated data structure, we proposed a dynamic authentication scheme for outsourcing data stream.
- The proposed authentication scheme supports the data stream dynamically increased, quickly updated and real-time verified. By introducing the multiple-collision trapdoor hash function on traditional Merkle Hash Tree (MHT), we can make the process of constructing authenticated structure of the data stream into two phases: off-line and on-line stages. There are three advantages for this. The first, we can maintain the cost of constructing for authenticated tree during off-line phase, which greatly improve the efficiency of constructing for the authenticated data structure in on-line phase. Second, when the data stream is being increased and updated, re-computing of all internal nodes and the root node is not necessary. Third, we can authenticate the identity of the data owners with the trapdoor information, without signature operation on root, thereby improving the efficiency of authentication. In addition, we can verify the order of the elements in a data stream, and prevent changing the order of the elements to change content.
- Our scheme supports attribute-based fine-grained authorisation verification. Because the authentication tree based on multiple-collision trapdoor hash function, when the data stream increase and update, its root values are relatively fixed, so we can use the CP-ABE algorithm to encrypt the root, implementing attribute-based fine-grained authorisation verification, enabling better control verifiable range, while reducing the complexity of key management.
- In this paper, we verified the scheme's correctness, verifiable and access security. Through the experiments we analysed and compared the performance and efficiency of the proposed scheme. The experimental results show that our scheme is efficient and feasible for outsourcing data stream.

1.2 Related work

Traditional cryptographic technologies can be used to solve the problem of data integrity and availability, but does not apply to verify the outsourcing the data stream [1, 2]. In addition, due to the expensive communication and computation, it is not a practical solution for verifying the data stream, after downloading the data stream. So the theory of authenticated data structures starts to draw public's attention these years. Early authenticated data structures were to solve the problem of certificate revocation in public key infrastructure. It focused on the certification dictionary and the execution of user queries. Later, authenticated data structures are widely studied for data storing, modification and searching under remote untrusted environment.

In 2003, Tamassia [3] first formally describes authenticated data structures model, which are essentially a computing model containing an objective structured collection s and three participants: data owner, user (inquirers) and responder. Among them, the user initiates a query for a data block, then, untrusted responder on behalf of the trust data owners makes response to the query. At the same time, the untrusted responder provides proof of the correctness of the response to the user.

MHT is one of the earliest authenticated data structures [4, 5]. MHT is a binary hash tree, leaf node is the hash value of each data block and intermediate nodes are the hash value of all child hash value after the connection. When validating data, you just need to check whether the hash value of root node are changed, and then you can determine whether the data block with corresponding leaf node are changed. In recent years, in order to solve the problem of data dynamic operation, and improve the authentication efficiency, people launched a series of studies [6–10] that aim at the weakness of the MHT itself. For example, Ling *et al.* [6] introduced the model of MHT into the B+ tree and present a MB + Tree, which organised and managed the outsourcing data, and reflected dynamic operation on data through the changes of the tree structure. Marek *et al.* [7] proposed skewed MHTs, in order to extend the MHT traversal algorithms without additional time and memory resources. They showed that extra nodes can be appended to a balanced MHT without changing the time and space bounds of the traversal algorithm. In order to improve playback performance for streaming content, Kandappu *et al.* [8] develop a new unbalanced authentication tree structure called the α-leaf tree that was a generalisation of the MHT. The α-leaf tree can reduce start-up delays for streaming applications by optimally readjusting the burden of authentication across packets.

Recently, Schroeder and Schroeder [11] improved the MHT from another angle, introducing the chameleon hash function [12] to construct a chameleon authenticated tree (CAT) that constructs the verification data stream scheme. The scheme advantage is based on CAT that has no fixed leaf nodes in advance. For clients who hold the trapdoor information, they can authenticate the new elements without computing in advance or recalculating other leaf node. The data in the setting stage of the data stream protocol can be unknown. Based on inspiration of the CAT scheme, we construct a dynamic authenticated structure with access control on outsourced data stream that extends the basis of the CAT. In order to prevent attacks the trapdoor collision can be used only once in the CAT. Therefore when updating data, it needs to rebuild the whole tree and validate the public key. Unlike the CAT, our scheme uses the trapdoor hash function without key exposure, which can compute multiple collisions on the same node and no need to update the value of the root node and public key, when updating data. Through the amortisation mechanism, the construction of authentication tree can be divided into off-line and on-line phases, which greatly improves the constructing efficiency of data authentication structures when the data stream increase and update. In addition, our scheme has fine-grained access control functions. It is neither based on symmetric key authentication, nor open to public validation of all users. It is only verified by users having a certain access strategy.

2 Preliminaries

2.1 Trapdoor Hash function

Definition 1: [12] A trapdoor hash function, $\widetilde{TH}$, is a tuple (I, H) whose components are defined as follows:

- I is a probabilistic polynomial time key generation algorithm that on input 1^k, outputs a pair hash/trapdoor key (HK, TK), such that the sizes of HK, TK are polynomial related to k.
- H is a family of randomised hash functions. Every hash function in H is associated with a hash key HK, and is applied to a message from a space M and a random element from a finite space R. The output of the hash function $\widetilde{TH}_{HK}$ does not depend on TK.

A trapdoor hash function $\widetilde{TH}$ is a trapdoor one-way hash function with some special properties, which prevents everyone except the holder of the trapdoor information from computing the collisions for a randomly given input. A trapdoor hash function satisfies the effective calculation collision resistance and trapdoor collision. The secure proof is given in [12].

Definition 2: [13] a multiple-collision trapdoor hash function, MTH is a tuple (I, I', H) whose components are defined as follows:

- I is a probabilistic polynomial time in the key generation algorithm, input 1^k, outputs a long-term hash/trapdoor key pair (HK, TK), such that the sizes of HK, TK are polynomial related to k. Note that (HK, TK) is associated with the all trapdoor hash functions in the family and can be used repeatedly during its lifespan.
- I' is a probabilistic polynomial time in the key generation algorithm, input 1^k, outputs a one-time hash/trapdoor key pair (HK', TK'), such that the sizes of HK', TK' are polynomial related to k. Note that (HK', TK') can be used only once during its lifespan.
- H is a random *hash* function family. Input 1^k, a pair hash/trapdoor key (HK, TK), a pair $(m, r) \in M \times R$ and a message $m' \neq m$, outputs a collision parameters r' and HK' such that $MTH_{HK}(m, r) = MTH_{HK'}(m', r')$. When $HK \neq HK'$, the hash key HK' along with the corresponding trapdoor key TK' is called a one-time key pair and can be used only once.

A multiple-collision trapdoor hash function MTH satisfies the five properties: effective calculation, trapdoor collision, collision resistance, key exposure resistance and semantic security. The secure proof is given in [13].

2.2 Ciphertext-policy attribute-based encryption

Definition 3: A CP-ABE scheme [14, 15] consists of four algorithms: ABE.setup, ABE.keygen, ABE.encrypt, and ABE.decrypt.

- $ABE.setup(1^k)$: Takes a security parameter 1^k, outputs a master public/secret key pair(pk, sk).
- $ABE.keygen(sk, A)$: Takes the master secret key sk and an attribute set $A \subseteq \Omega$ where Ω is the attribute universe, outputs a decryption key dkA.
- $ABE.encrypt(m, \omega, pk)$: Takes the master public key pk, a message m and an access policy ω, produces a ciphertext c.
- $ABE.decrypt(c, dk, \omega)$: Takes the master public key pk, a decryption key dkA and a ciphertext c, outputs a message m if A satisfies ω which is associated with c.

The definition of full security of CP-ABE is given in [14, 15].

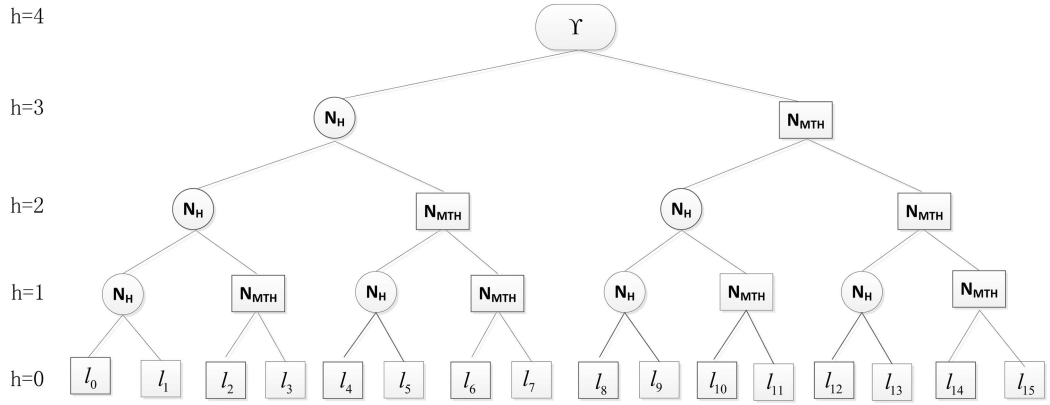

Fig. 1 *AC-MTAT*

3 The dynamic authenticated data structure with access control

3.1 Overview

The dynamic authenticated data structure with access control is essentially the multiple-collision trapdoor hash authentication tree (MTAT) with access control that can be used to build efficient outsourcing data stream authentication mechanisms and overcome the aforementioned challenges. Comparing with the traditional MHT and CAT, the dynamic authenticated data structure with access control, which we named AC-MTAT, is a binary tree composed of CP-ABE algorithm and multiple-collision trapdoor hash function. The root of the tree can only be checked and verified by the client with satisfied access strategy. Fig. 1 illustrates an example AC-MTAT: The depth of the tree is defined by D and the level of a node in the tree is denoted by $h = 0, \ldots, D-1$, where leaf nodes have level $h = 0$ and the root node in turn has level $h = D-1$. each non-leaf node N will have a maximum of two child nodes, each right-handed node of the tree is represented as N_{MTH} whose hash value is computed by a multiple-collision trapdoor hash function, each left-handed nodes is represented as N_{H} whose hash value is computed by a collision-resistant hash function; the leaf nodes is represented as l whose hash value is computed by a trapdoor hash function; the root node ρ is computed by a multiple-collision trapdoor hash function and encrypted by CP-ABE, denoted Υ. Each node value of the specific algorithm we will give in Section 4.

3.2 Formal definition of AC-MTAT

Definition 4: A kind of dynamic authenticated structures with access control for outsourcing data stream is essentially an AC-MTAT, which is a tuple of PPT algorithm $AC-MTAT = (MtatGen, IkeyGen, CertIssue, InitiRoot, Add\_Leaf, Update\_Leaf, MtatVrfy)$:

- $(SK, VK) \leftarrow MtatGen(1^k, D)$: The AC-MTAT key generation algorithm takes as input a security parameter 1^k and an integer D that defines the depth of the tree. It returns a private key SK and verification key VK, then, it publishes VK and keeps SK as secret.
- $(sk_I, vk_I) \leftarrow IkeyGen(1^k)$: The IkeyGen generation algorithm takes as input a security parameter 1^k. It returns a public key pk_I and the corresponding secret key sk_I. It publishes pk_I and keeps sk_I as secret.
- $cert_A \leftarrow CertIssue(sk_I, pk_I, A)$: The cert generation algorithm takes as input a pair of key (sk_I, vk_I) and an attribute set A. It outputs an access certificate $cert_A$.
- $\Upsilon \leftarrow InitiRoot(pk_I, root, \omega)$: The initialisation algorithm takes as input a public key pk_I, a root of the authentication tree, and an access control structure ω. It outputs public information Υ related to root and ω.
- $(SK', i, aProof) \leftarrow Add\_Leaf(SK, l)$: The authentication proof generation algorithm takes as input a private key SK and a leaf $l \in L$ from some leaf space L. It outputs a key SK′, the index i of l in the tree, and the authentication proof *aProof*.
- $(SK', nPath', aPath') \leftarrow Update\_Leaf(SK, VK, i, l')$: The update algorithm takes as input a pair of key (SK, VK), a leaf $l' \in L$ and the index i of l' in the tree. It updates the i^{th} leaf nodes to l', updates the nodes that lay on the path $nPath'$ from the leaf l' to the root, and updates the nodes that lay on the authentication path $aPath'$ of l'.
- $\{0,1\} \leftarrow MtatVrfy(VK, i, l, aProof, \Upsilon)$: The verification algorithm takes as input a public key VK, an index i, a leaf l, an authentication proof *aProof*, and a root Υ. It outputs 1 iff l is the i^{th} leaf in the tree, otherwise 0.

4 Construction of the AC-MTAT

Let H be a collision-resistant hash function, $\widetilde{TH}$ is a rapdoor hash function, MTH is a multiple-collision trapdoor hash function, and CP-ABE is a ciphertext-policy attribute-based encryption scheme. We define the MTAT with access control $AC-MTAT = (MtatGen, IkeyGen, CertIssue, Add\_Leaf, Updata\_Leaf, MtatVrfy)$ as follows:

- *MtatGen*: Consists of the following key generation algorithm.

  (1) $ThKeyGen(1^k)$: The trapdoor hash key generation algorithm takes as input a security parameter 1^k, It returns a pair of trapdoor/hash keys $(tk, hk) \leftarrow \widetilde{TH}(1^k)$, publishes hk and keeps tk as secret.

  (2) $MthKeyGen(1^k, D)$: The multiple-collision trapdoor hash key generation algorithm takes as input a security parameter 1^k and an integer D that defines the depth of the tree. It returns a pair of long-term trapdoor/hash keys $(mtk, mhk) \leftarrow MKeyGen(1^k)$ and a pair of one-time trapdoor/hash keys $(\alpha, Y) \leftarrow MKeGen(1^k)$. In order to facilitate description, (α^t, Y^t) denotes the current pair of one-time trapdoor/hash keys, then denotes the (α^{t-1}, Y^{t-1}) last update pair of the trapdoor/hash keys, (α^{t+1}, Y^{t+1}) denotes the next update pair of the trapdoor/hash keys.
- $IkeyGen(1^k)$: Taking a security parameter 1^k as input, the IkeyGen generation algorithm runs $ABE.setup$ algorithm to generate a pair of keys $(sk_I, pk_I) \leftarrow ABE.setup(1^k)$, publishes pk_I and keeps sk_I as secret.
- $CertIssue(sk_I, pk_I, A)$: For any attribute set A, the CertIssue algorithm runs the $ABE.keygen$ algorithm to generate a decryption key. Then let $cert_A := dk_A$ be a certificate corresponding to A.
- $InitiRoot(pk_I, root, \omega)$: The InitiRoot algorithm computes a root ρ of the initially tree by using $Add\_Leaf(SK, l)$ algorithm, then

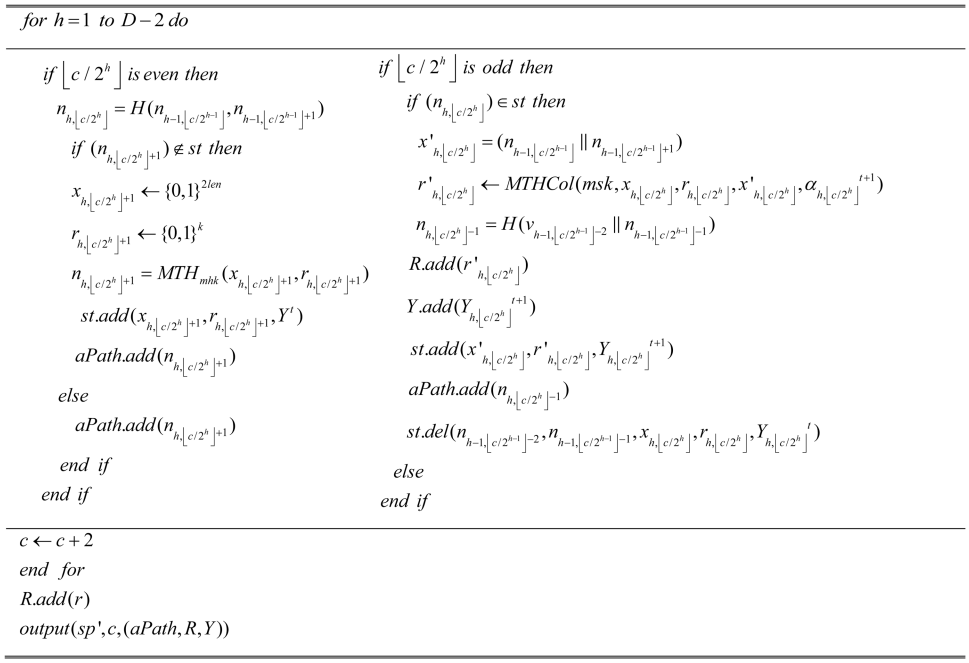

Fig. 2 *Algorithm Add_Leaf gets the counter c from the state st*

generates $\Upsilon \leftarrow ABE.encrypt(\rho, \omega, pk_I)$ by inputting an attribute structure ω and a root ρ. Thus only the clients, who have attributes satisfying ω, can decrypt Υ by using their certificates and then verify the leaf node by performing *MtatVrfy* algorithm .

- $Add\_Leaf(SK, l)$: The path generation algorithm is to perform the leaf node adding procedure in two phases. The first phase is performed off-line before the stream message is known and the second phase is performed on-line after the stream message is known. It sets the counter $c \leftarrow 0$ and the state $st \leftarrow (c, D, [(x, r, Y^t)])$. Then, it picks a random value r, sets $(l_c, l_{c+1}) \leftarrow \widetilde{TH}(l_c, r)$. Let $SK = (tk, mtk, st)$, $VK = (hk, mhk, \Upsilon)$, publishes VK and keeps SK as secret.

  I. Off-line:

  (1) The algorithm *Add_Leaf* picks random values l and $r \leftarrow \{0,1\}^k$, then computes $(l_0, l_1) \leftarrow \widetilde{TH}(l, r)$, $n_{1,0} = H(l_0 \| l_1)$. It choices at random $(x_{h,1}, r_{h,1}) \in_R M \times R$(for $h = 1, \ldots, D-2$), and computes the trapdoor hash value $n_{h,1} = MTH_{mhk}(x_{h,1}, r_{h,1})$(for $h = 1, \ldots, D-2$). Subsequently, it computes $n_{h,0} = H(n_{h-1,0} \| n_{h-1,1})$(for $h = 1, \ldots, D-2$) and $x_\rho \leftarrow (n_{D-2,0} \| n_{D-2,1})$. At last, it choices a random value r_ρ, computes $\rho = MTH_{mhk}(x_\rho, r_\rho)$ and stores it, where *mhk* is a long hash key, the corresponding long trapdoor key is *msk*.
  (2) It generates the verifiable root $\Upsilon \leftarrow ABE.encrypt(\rho, w, pk)$ by runs $ABE.encrypt$ algorithm.
  (3) It computes the authentication proof for l as defined in the algorithm *MtatVrfy* up to the level $D-2$. The algorithm computes the corresponding authentication path for the leaf l as $aPath = ((n_{h+1,1}, \ldots, n_{D-2,1}), R, Y)$, and sets $R = (r_\rho, r)$.
  (4) It sets the counter $c \leftarrow 2$, and the state $st' \leftarrow (c, D, x_\rho, r_\rho, [x, r, Y_i], l_0, l_1)$. The algorithm returns $SK' = (tk, mtk, st')$, the index 0, and the authentication path *aPath*.
  II. On-line

The algorithm *Add_Leaf* gets the counter c from the state st. If $c \succ 0$, then it runs the $Add\_Leaf(on-line)$ algorithm, and creates a new list authentication proof proceeds as shown in Fig. 2.

- $Update\_Leaf(SK, VK, i, l')$:

  (1) The update algorithm sets $l \leftarrow l'$, and computes $(l_i, l_{i+1}) \leftarrow \widetilde{TH}(l_i, r_i)$.
  (2) The update algorithm updates the parent nodes of on the l' as shown in Fig. 3.
- $MtatVrfy(VK, i, l, a \text{ Pr } oof, \Upsilon)$:

  (1) The verifier computers $\rho*$ as $\rho* \leftarrow ABE.decrypt(cert_A, pk_I, \Upsilon)$.
  (2) The verification algorithm sets $(l_i, l_{i+1}) \leftarrow \widetilde{TH}(l_i, r)$ and computes the node $n_{h,i}$ for $h = 2, \ldots, D-2$ as follows:

$$\text{If } \lfloor i/2^h \rfloor \equiv 1 \mod 2$$
$$x \leftarrow n_{h-1,\lfloor i/2^{h-1} \rfloor} \| n_{h-1,\lfloor i/2^{h-1} \rfloor+1}$$
$$n_{h,\lfloor i/2^h \rfloor} \leftarrow MTH_Y(x; r_{h,\lfloor i/2^h \rfloor}), \text{ with } r_{h,\lfloor i/2^h \rfloor} \in R$$

$$\text{If } \lfloor i/2^h \rfloor \equiv 0 \mod 2$$
$$x \leftarrow n_{h-1,\lfloor i/2^{h-1} \rfloor-2} \| n_{h-1,\lfloor i/2^{h-1} \rfloor-1}$$
$$n_{h,\lfloor i/2^h \rfloor} \leftarrow H(x)$$

  (3) The verifier query the index i and obtain the authentication proof $aProof = (aPath, R, Y)$, and the authentication proof *aProof* query proceeds as shown in Fig. 4.
  (4) The verifier computers the root node $\hat{\rho}$ as $\hat{\rho} \leftarrow MTH_Y(n_{D-2,0} \| n_{D-2,1}; r_\rho)$(with $r_\rho \in R$), by inputting the public VK, index i of the leaf l, the node $n_{h,i}$ (for $h = 2, \ldots, D-2$) and the authentication proof $aProof = (aPath, R, Y)$, where R and Y is a non-empty set that contains all randomness and one-time hash key that are necessary to compute the trapdoor hash functions. If $\hat{\rho} = \rho*$ then the leaf is authenticated, and otherwise rejected.

5 Security analysis

In this section, we will analyse the security of our AC-MTAT construction in detail, including Correctness, Verifiability and Access Security.

for $h=1$ *to* $D-2$ *do*

if $\lfloor i/2 \rfloor$ *is even then*

$n_{h,\lfloor i/2^h \rfloor} = H(n_{h-1,\lfloor i/2^{h-1} \rfloor} \| n_{h-1,\lfloor i/2^{h-1} \rfloor+1})$

$x'_{\rho} \leftarrow n_{D-2,0} \| n_{D-2,1}$

$r'_{\rho} \leftarrow MTHCol(\alpha_{\rho}^{t}, x_{\rho}, r_{\rho}, x'_{\rho}, \alpha_{\rho}^{t+1})$

$st.add(x'_{\rho}, r'_{\rho}, Y_{\rho}^{t+1})$

$R.add(r'_{\rho})$

$Y.add(Y_{\rho}^{t+1})$

$st.del(x_{\rho}, r_{\rho}, Y_{\rho}^{t})$

end if

if $\lfloor i/2 \rfloor$ *is odd then*

$x'_{h,\lfloor i/2^h \rfloor} = (n_{h-1,\lfloor i/2^{h-1} \rfloor} \| n_{h-1,\lfloor i/2^{h-1} \rfloor+1})$

$r'_{h,\lfloor i/2^h \rfloor} \leftarrow MTHCol(\alpha_{h,\lfloor i/2^h \rfloor}^{t}, x_{h,\lfloor i/2^h \rfloor}, r_{h,\lfloor i/2^h \rfloor}, x'_{h,\lfloor i/2^h \rfloor}, \alpha_{h,\lfloor i/2^h \rfloor}^{t+1})$

$R.add(r'_{h,\lfloor i/2^h \rfloor})$

$Y.add(Y_{h,\lfloor i/2^h \rfloor}^{t+1})$

$st.add(x'_{h,\lfloor i/2^h \rfloor}, r'_{h,\lfloor i/2^h \rfloor}, Y_{h,\lfloor i/2^h \rfloor}^{t+1})$

$st.del(n_{h-1,\lfloor i/2^{h-1} \rfloor}, n_{h-1,\lfloor i/2^{h-1} \rfloor+1}, x_{h,\lfloor i/2^h \rfloor}, r_{h,\lfloor i/2^h \rfloor}, Y_{h,\lfloor i/2^h \rfloor}^{t})$

end if

end for

Fig. 3 *Update algorithm updates the parent nodes of on the l′*

for $h=1$ *to* $D-2$ *do*

if $\lfloor i/2 \rfloor$ *is even then*

for $h=1$ *to* $D-2$

$aPath = aPath.add(n_{h,\lfloor i/2^h \rfloor+1})$

$R = R.add(R_{h,\lfloor i/2^h \rfloor+1})$

$Y = Y.add(R_{h,\lfloor i/2^h \rfloor+1})$

end if

if $\lfloor i/2 \rfloor$ *is odd then*

for $h=1$ *to* $D-2$

$aPath = aPath.add(n_{h,\lfloor i/2^h \rfloor-1})$

$R = R.add(R_{h,\lfloor i/2^h \rfloor-1})$

$Y = Y.add(R_{h,\lfloor i/2^h \rfloor-1})$

end if

$aProof = (aPath, R, Y)$

end for

$output(aProof)$

Fig. 4 *Algorithm to generate the authentication proof*

5.1 Correctness

When both the client and the server are honest, the output of the scheme will be $\overline{\rho} \leftarrow MTH_Y(n_{D-2,0} \| n_{D-2,1}; r_{\rho})$ (with $r_{\rho} \in R$). Concretely, if a client whose attribute set A satisfies the access structure w, then it can decrypt root Υ by performing $\rho \leftarrow ABE.decrypt(cert_A, pk_w)$ in Query. Then by comparing $\overline{\rho}$ and ρ, if $\overline{\rho} = \rho$, then the leaf is authenticated, and otherwise rejected. Hence, the correctness of this scheme depends on that of the based the one-way of the multiple-collision trapdoor hash function and CP-ABE is fully secure.

5.2 Verifiability

5.2.1 Secure model: The AC_MTAT is security depend on the security of the MTAT and CP-ABE. In the AC-MTAT, only the clients who satisfy the permissions can parse root Υ to ρ and verify the data, that is the adversary cannot computer a valid ρ. In the following, we will prove that our construction for AC-MTAT satisfies Verifiability and Access Security.

We suppose that the adversary can break the security of the CP-ABE, that is, the adversary can learn ρ by using $\rho \leftarrow Decrypt(\Upsilon, \omega)$, so we give the verifiability define of AC-MTAT.

Definition 5: The AC-MTAT is verifiability if it satisfies the following properties:

- An adversary should not be able to change the structure of the MTAT.
- An adversary should not be able to change the sequence of the leaves.
- An adversary should not be able to substitute any leaf.
- An adversary should not be able to add further leaves to a MTAT.

We formalise above the property by an interactive game $G_A^{AC-MTAT}(1^k)$ between the challenger and adversary A, where 1^k is a secure parameter. The secure model is based on the CAT secure model [11]. The challenger generates a key pair (SK, PK) and hands the verification key PK over to the adversary A. The adversary may send z leaves $l_1, \ldots, l_{z(k)}$ (adaptively) to the challenger who returns the corresponding authentication proof $aProof_1, \ldots, aProof_{z(k)}$. Afterwards, the adversary tries to break the structure of the MTAT by outputting a leaf that has not been added to the tree at a particular position. More formally:

Setup: The challenger runs the algorithm $MtatGen(1^k, D)$ to compute a private key SK and a verification key VK. The adversary A is given VK.

Insert queries: Proceeding adaptively, the attacker A streams a leaf $l \in L$ to the challenger. The challenger picks a value $r \leftarrow \{0,1\}^k$ uniformly at random, sets $x \leftarrow \widetilde{Th}(l, r)$, computes $(sk', i, aProof) \leftarrow Add\_Leaf(sk, l)$ and returns $(i, aProof)$ to A. Denote by $Q := \{(l_1, 1, aProof_1), \ldots, (l_{z(k)}, z(k), aProof_{z(k)})\}$ the ordered sequence of query-answer pairs. In addition, it records the pair (l, r) in the table T.

Update queries: Proceeding adaptively, the attacker A updates queries of the form $(i, l'_n)(1 \le n \le z)$ to the challenger. The challenger picks a value $r \leftarrow \{0,1\}^k$ uniformly at random, sets $x' \leftarrow \widetilde{Th}(l', r)$, computes $(sk', nPath', aProof') \leftarrow Updata\_Leaf(sk, i, x')$, where $aProof' = (aPath', R', Y')$ and returns $(i, aProof')$ to A. Denote by $Q := \{(l_1, i, aProof_1), \ldots, (l_{z(k)}, i, aProof_{z(k)})\}$ the ordered sequence of query-answer pairs.

Output: Eventually, if one of the following is true, the attacker is said to win the game.

Type-1: A outputs x_i^*, and $x_i^* = x_i$,where x_i was initially stored at the position i of the MTAT.

Type-2: A outputs $(l*, i*, aProof*)$, $1 \le i* \le z(k)$ and $(l*, i*, aProof*) \notin Q$ and $MtatVrfy(vk, i*, l*, aProof*) = 1$.
Type-3: A outputs $(l*, i*, aProof*)$, $z(k) \prec i* \le n$ and $MtatVrfy(vk, i*, l*, aProof*) = 1$.

The advantage of the adversary, $Adv_A^{AC-MTAT}$, is the probability that the adversary A wins in the above game.

Definition 6: An AC-MTAT, defined by the efficient algorithms $AC-MTAT = (MtatGen, IkeyGen, CertIssue, InitiRoot, Add\_Leaf, Updata\_Leaf, MtatVrfy)$ with n leaves, is verifiability security if for any $z \in N$, and for any PPT algorithm A, the probability $adv_A^{AC-MTAT}$ is negligible (as a function of k).

Theorem 1: If H is a collision-resistant hash function, $\widetilde{TH}$ is a one-way collision-resistant trapdoor hash function, and MTH is a one-way multiple-collision trapdoor hash function without key exposure, then our construction for AC-MTAT satisfies verifiability.

Proof: The proof follows the one of [11]. Suppose there is an adversary A win the game in Section 5.2.1 with non-negligible probability, then A can output Type-1, Type-2 or Type-3. If A outputs Type-1, this implies that there is a trapdoor collision about leaf l, then there is $\widetilde{TH}(l_i^*, r_i^*) = x_i^* = x_i = \widetilde{TH}(l_i, r_i)$. However, this contradicts the assumption that $\widetilde{TH}$ is collision-resistant. If A outputs Type-2 or Type-3, this implies that A can forgery $aProof*$ of $l*$ in the $i*$. In the following, we will distinguish differences between the two cases.
Type-2: When A outputs a Type-2 forgery, that is A returns a tuple $(l*, i*, aProof*)$ with $1 \le i* \prec z$. This implies we constructed an algorithm B against the collision-resistant in either MTH or H. It breaks the multiple-collision trapdoor hash function or the collision-resistance of the hash function as follows:
Setup: The algorithm B picks z values $l_1, \ldots, l_z$ uniformly at random from $\{0,1\}^k$ and z strings at random from $\{0,1\}^k$. It stores these values in a table T and generates a key pair of the trapdoor hash $(tk, hk) \leftarrow THKeyGen(1^k)$ and outputs $x_i \leftarrow \widetilde{Th}(l_i; r_i)$ for $i = 1, \ldots, z$. It then picks dummy node $n_i \leftarrow \{0,1\}^{len}$ uniformly at random such that the entire tree has depth $D = poly(k)$ and computes the root ρ of tree from the bottom to the top. Furthermore, the algorithm B computes the authentication proof $((l_1, aProof_1), \ldots, (l_z, aProof_z))$ and sets $VK \leftarrow (th, mhk, \rho)$ and runs a black-box simulation of A on VK.
Insert query: Whenever A wish to add a leaf l to the tree, it picks a fresh value r uniformly at random, sets $x \leftarrow \widetilde{Th}(l; r)$, computes the authentication proof as $(SK', i, aProof) \leftarrow addLeaf(sk, x)$, and returns $aProof = (aPath, i, r)$, In addition, it records the pair (l, r) in a table T.
Update query: Whenever A wish to update l of the ith leaf node to l'. The algorithm B answers update queries of the form (i, l') in the following way. It computes a collision in the multiple-collision trapdoor hash $r' \leftarrow Col(\alpha^t, l, r, l', \alpha^{t+1})$ and updates the ith leaf node l to l', r to r' and stores the one-time hash key Y^{t+1}. Then, it updates the nodes that lay on the authentication path $aPath'$ of l' and updates the nodes that lay on the path $nPath'$ from the leaf l' to the root. It returns the $aProof = (aPath, i, r, Y^{t+1})$ to A.
Output: Eventually, A stop**s**, outputting a pair $(l*, i*, aProof*) \notin Q$ with $aProof* = (aPath*, R*, Y*)$, $x* \leftarrow \widetilde{Th}(l*; r*)$. The algorithm B returns $(x*, i*, aProof*)$.
Since A runs in polynomial-time, the algorithm B is efficient. This implies there exists a hash collision $H(aPath_i \parallel nPath_i) = H(aPath*_i \parallel nPath*_i)$ or a trapdoor hash collision $MTH(aPath_i \parallel nPath_i) = MTH(aPath*_i \parallel nPath*_i)$ based on [11], where $nPath*_i = (v^*_{1,\lfloor i/2 \rfloor}, \ldots, V^*_{D-2,\lfloor i/2D-2 \rfloor})$. This contradicts the assumption that MTH or H is collision-resistant.

Type-3: When A outputs a Type-3 forgery, that is A returns a tuple $(l*, i*, a\ \mathrm{Pr}\ oof*)$ with $z+1 \le i* \prec 2^D$. This implies we construct an algorithm B against either the collision-resistant of MTH or the one-wayness of MTH.
The main observation is that the authentication path corresponding to A's output, must contain at least one right-handed node on the authentication path of the last inserted value l_z. Since it must be a right-handed node, it is computed by a multiple-collision trapdoor hash function. Furthermore, the update queries do not change this fact as they only involve 'old' values. We build an algorithm B that either breaks the one-way or the collision-resistance of MTH. We describe the reduction to the one-wayness, but we stress that the one against collision-resistance is roughly the same.
Setup: The algorithm B runs in a black-box and returns z leaves $l_1, \ldots, l_z$ to A. It picks t trapdoor hash values $n_1, \ldots, n_j^*, \ldots, n_t$ as dummy nodes, where $n_j^* \leftarrow MTH_Y(x_j^*; r_j^*)$. Then, B computes the tree from the bottom to the top using where the first z leaves are $l_1, \ldots, l_z$ and uses the t dummy nodes $n_1, \ldots, n_j^*, \ldots, n_t$ such that the entire tree has depth $D = poly(k)$ and let ρ be the resulting root node.
Insert query: Whenever A wish to add a leaf l' to the tree, then B retrieves the pair (x_i, r_i) from T and computes a collision in the multiple-collision trapdoor hash for which B knows the trapdoor $r' \leftarrow Col(mtk, l, r, l', Y^t)$. It returns the corresponding authentication proof $aProof = (aPath, i, r')$ to A.
Update query: At some point, A sends (i, l') to the challenger to update the i^{th} leaf node to l'. The algorithm B retrieves the pair (x_i, r_i, Y^t) from T and computes a collision in the multiple-collision trapdoor hash for which B knows the trapdoor $r' \leftarrow Col(Y^t, l, r, l', Y^{t+1})$. Then, it updates the nodes that lay on the authentication path aPath′ of l' and stores the one-time hash key $Y^t \leftarrow Y^{t+1}$ in the table, then $aProof = (aPath, i, r, Y^t)$, and returns the $aProof_i$ to A.
Output: Eventually, A stops, outputting a tuple $(l*, i*, aProof*)$ with $x* = \widetilde{Th}(l*; r*)$, such that $MtatVrfy(PK, i*, l*, aProof*) = 1$ and $z+1 \le i* \prec 2^D$.
For the analysis, observe that A outputs a tuple $(l*, i*, aProof*)$, where $aProof* = (aPath*, i*, R*, Y*)$. Let $nPath* = (v^*_{1,i/2}, \ldots, v^*_{D-2,i/2D-2})$ denote the path from the leaf $l*$ to the root. Because$z+1 \le i* \prec 2^D$, then at least one node of $n_1, \ldots, n_j^*, \ldots, n_t$ lies on the authentication path of $l*$. Since t is polynomially bounded, B can guess the index with non-negligible probability. Assuming that B guessed the index correctly, then, there exists an index j such that $n'_j = n_j^*$. This implies B can obtain a pair (x_j^*, r_j^*) by using $n_j^* \leftarrow MTH_Y(x_j^*, r_j^*)$ such that $n'_j \leftarrow MTH_Y(x_j^*, r_j^*)$. If $(x_j^*, r_j^*) = (x'_j, r'_j)$, then B breaks the one-wayness of MTH, If $(x_j^*, r_j^*) \ne (x'_j, r'_j)$, then B breaks the collision-resistant or one-time collision-resistant of MTH. Furthermore, assume that A succeeds with non-negligible probability $\varepsilon(k)$ then B succeeds with non-negligible probability $\varepsilon(k)/t$. □

5.3 Access security

Proof in AC-MTAT scheme, only the client with satisfied access strategy can check and verify outsourcing data stream. In our construction, to outsourcing data stream, the client has to decrypt root Υ and receive proof from the AC-MTAT. We assume that the proof from the AC-MTAT is secured such that if and only if a client learns the ρ, it can check and verify the outsourcing data stream. Thus, we formalise above the process by an interactive game $G_A^{Access}(1^k)$ between the challenger and adversary A, where 1^k is a secure parameter. More formally:

Setup: The challenger runs the algorithm $IskeyGen(1^k)$ to compute a private key sk_I and a verification keypk_I. The adversary A is given pk_I. $\Upsilon \leftarrow Initialisation(k, pk_I, \rho, \omega)$.

Queries: Proceeding adaptively, the attacker A sends a sequence of queries $\omega_1, \omega_2, \ldots, \omega_q$ (adaptively) to the challenger who returns the corresponding authentication keys $cert_{\omega 1}, \ldots, cert_{\omega q}$. The challenger computes $\rho_i \leftarrow Decrypt(cert_{\omega_i}, \Upsilon)$ and returns ρ_i to A. Denote by $Ce := \{(cert_{\omega 1}, \rho_1), \ldots, (cert_{\omega q}, \rho_q)\}$ the ordered sequence of query-answer pairs.

Output: Eventually, A stops, outputting a pair $(cert, \rho)$ and $MTAT.Verify(auth, i, l, pk, \rho) = 1$, then the attack is said to win the game.

We define adv_A^{Access} to be the probability that the adversary A wins in the above game.

Definition 6: A multiple-collision trapdoor authentication tree with access control is access security, for any PPT algorithm A, the probability $adv_A^{AC-MTAT}$ is negligible.

Theorem 2: If the AC-MTAT scheme is verifiability and CP-ABE scheme is fully secure, then our construction for AC-MTAT satisfies access security.

Proof: If there is an adversary A who can win the game $G_A^{Access}(1^k)$, then there exists either the adversary A who can break AC-MTAT's verifiability with non-negligible probability, or the adversary A who can break CP-ABE full security with non-negligible probability. Theorem 1 proves that the AC-MTAT satisfies verifiability. Based on Theorem 1, if the CP-ABE is full security, then AC-MTAT satisfies access security.

There exists an adversary A who can win the game G_A^{Access} with non-negligible probability, then we can construct an algorithm B who can break the full security of CP-ABE.

Setup: B is given the public key pk_I of CP-ABE. Then B generates Υ by running $Initialisation(k, pk_I, \rho, \omega)$ for a client and an attribute set ω, where $\rho = Decrypt_{ABE}(cert_A, \Upsilon)$, and sends (pk_I, Υ) to A.

Queries: When A make queries for some attribute A_i where A_i does not satisfy ω, B uses its own oracle to answer A. B selects a random value t, and sends (t, ρ) to the challenger. After receiving the answer C, B sets C as the answer.

Output: Finally, if A's output is ρ, and $MTAT.Verify(aProof, i, l, pk, \rho) = 1$, then B outputs '1', else B outputs '0'.

Because the based CP-ABE is fully secure, so there is not an adversary A can output ρ with non-negligible probability, then for any adversary A, we have $\Pr[G_A^{Access} = 1] \leq negli(k)$.□

6 Performance analysis

The efficiency of this scheme is mainly manifested in computing the hash function, the multiple-collision trapdoor hash function, and the trapdoor collision. Therefore, we mainly use the above three indicators to analyse the efficiency of this scheme. We let $Exp(H)$ denote the time to compute a hash function, let $Exp(MTH)$ denote the time to compute a multiple-collision trapdoor hash function, let $Exp(Col)$ denote the time to compute a trapdoor collision. Our construction can be instantiated with any multiple-collision trapdoor hash function. But in order to facilitate evaluation and comparison, our scheme adopts the multiple-collision trapdoor hash function [13] which is secure under the discrete logarithm assumption that base on security. Furthermore, as same as the KR chameleon hash function, the multiple-collision trapdoor hash function supports batch verification techniques which are proposed by Bellare *et al.* [16].

Table 1 Time comparison of four experiments

	Time of initial authentication tree, ms	Time of append leaf node, ms	Time of update leaf node, ms	Time of verify leaf node, ms
10 (level)	30	7–13	7–21	22–26
20 (level)	53	7–21	7–29	31–32
40 (level)	101	8–38	7–45	49–53
80 (level)	194	8–72	7–86	82–83

The design goal of our scheme is to improve the efficiency of dynamic appending, update and verification on outsourced data stream. So, we will analysis the performance efficiency from these three aspects.

i. The efficiency analysis of dynamic appending on outsourced data stream

Due to the multiple-collision trapdoor such that the new leaf authenticates under the same root without key exposure, we can construct an initial authentication tree based on amortisation mechanism in the off-line. According to our algorithm, when increase the leaf node, the scheme is divided into two cases: if insert the left leaf node, computing time is at most about $(D-3)Exp(H) + (D-2)Exp(MTH) + Exp(Col)$; if insert the right leaf node, computing time is at most about $Exp(MTH) + Exp(Col)$.

ii. The efficiency analysis of real-time update on outsourced data stream

Similarly, the process of updating leaf node is divided into two cases: if update left leaf node, computing time is at most about $(D-3)Exp(H) + Exp(Col)$; if update right leaf node, computing time is $Exp(Col)$. CAT scheme in data updating need to rebuilt the authentication tree and publish a new public key. Obviously the complexity is higher and time delay is longer.

iii. The efficiency analysis of verification on outsourced data stream

Without considering access control, the efficiency of verification is same as the CAT. In the worst cases, the scheme need to compute D times trapdoor hash function, which computing time is $D(Exp(MTH))$.

We have implemented some important components of the AC-MTAT on the Intel Core i5 using 4GB 1600 MHz DDR3 RAM. The code was written in JAVA 1.6 and all cryptographic operations use the Java security package. That is, we have implemented the authentication tree by implementing trapdoor hash function in [13] and MD5 hash function.

As shown in Table 1, we carried out four groups of experiments, which we executed each algorithm 500 times on the AC-MTAT with depth from 10 to 80. The first experiment is an evaluation of average computational costs time to construct an initial authentication tree. The second experiment is an evaluation of average computational costs time of dynamic appending on outsourced data stream in both best case and worst circumstance. The third experiment is an evaluation average computational costs time of dynamically update on outsourced data stream in both best and worst circumstances. The fourth experiment is an evaluation average computational costs time of verification on outsourced data stream in both best and worst circumstances. The evaluation results show that our scheme is effective and practical.

7 Conclusion and future work

We have introduced the AC-MTAT, an authenticated data structure, which is suitable for the outsourcing data stream. The AC-MTAT supports to dynamic appending, real-time update, efficient verification and fine-grained access control on outsourced data stream. We proved the correctness, verifiability and access security of the AC-MTAT. We have analysed the performance efficiency and also implemented a prototype of AC-MTAT to evaluate the computational costs time. The experimental results show that our scheme is effective and practical on outsourced data stream. In

addition, the AC-MTAT scheme can be used in many applications such as cloud storage, cloud auditing and outsourcing data exchange, etc. In the next step, we will research the outsourcing data stream exchange scheme and auditing scheme based on the AC-MTAT.

8 References

[1] Hsiao, H.-C., Lin, Y.-H., Studer, A., *et al.*: 'A study of user-friendly Hash comparison schemes'. Proc. Ann. Computer Security Applications Conf. (ACSAC), 2009, pp. 105–114

[2] Zhu, Y., Wang, H., Hu, Z., *et al.*: 'Efficient provable data possession for hybrid clouds'. Proc. 17th ACM Conf. Computer and Communications Security, 2010, pp. 756–758

[3] Tamassia, R.: 'Authenticated data structures'. Proc. 11th Annual European Symp., 2003 (LNCS, **2832**), pp. 2–5

[4] Merkle, R.C.: 'Protocols for public key cryptosystems'. Proc. Symp. on Security and Privacy, 1980, pp. 122–134

[5] Merkle, R.C.: 'A certified digital signature'. Proc. CRYPTO'89, 1990 (LNCS, **435**), pp. 218–238

[6] Ling, L.: 'Reserch on some issues of data security in cloud computing services'. *PhD thesis*, University of Science and Technology of China, 2012

[7] Marek, K., Yakov, N.: 'A note on traversing skew Merkle trees'. ECCC, 2004

[8] Kandappu, T., Sivaraman, V., Boreli, R.: 'A novel unbalanced tree structure for low-cost authentication of streaming content on mobile and sensor devices'. 2012 Ninth Annual IEEE Communications Society Conf. on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012, pp. 488–496

[9] Liu, C., Chen, J., Yang, L., *et al.*: 'Authorized public auditing of dynamic big data storage on cloud with efficient verifiable fine-grained updates', *IEEE Trans. Parallel Distrib. Syst.*, Early Access Online, 2013, doi: 10.1109/TPDS.2013.191

[10] Tang, Y., Wang, T., Hu, X., *et al.*: 'Outsourcing multi-version key-value stores with verifiable data freshness'. Presented at the 2014 IEEE 30th Int. Conf. on Data Engineering (ICDE), 2014, pp. 1214–1217, doi: 10.1109/ICDE.2014.6816744

[11] Schroeder, D., Schroeder, H.: 'Verifiable data streaming'. Proc. 2012 ACM Conf. on Computer and Communications Security, 2012, pp. 953–964, doi: 10.1145/2382196.2382297

[12] Ateniese, G., de Medeiros, B.: 'On the key exposure problem in chameleon hashes'. SCN 2004, 2005 (LNCS, **3352**), pp. 165–179

[13] Chandrasekhar, S., Chakrabarti, S., Singhal, M.: 'A trapdoor Hash-based mechanism for stream authentication', *IEEE Trans. Dependable Secur. Comput.*, 2012, **9**, (5), pp. 699–713

[14] Bethencourt, J., Sahai, A., Waters, B.: 'Ciphertext-policy attribute-based encryption'. IEEE Symp. on Security and Privacy, 2007, pp. 321–334

[15] Lewko, A., Okamoto, T., Sahai, A., *et al.*: 'Fully secure functional encryption: attribute-based encryption and (hierarchical) inner product encryption'. EUROCRYPT 2011, 2010 (LNCS, **6110**), pp. 62–91

[16] Bellare, M., Garay, J.A., Rabin, T.: 'Fast batch verification for modular exponentiation and digital signatures'. Advances in Cryptology – EUROCRYPT'98, Espoo, Finland, 31 May–4 June 1998 (LNCS, **1403**), pp. 236–250