SECProv: Trustworthy and Efficient Provenance Management in the Cloud

Shams Zawoad, Ragib Hasan, and Kamrul Islam
Department of Computer and Information Sciences
University of Alabama at Birmingham, AL 35294, USA
szawoad42@gmail.com, ragib@uab.edu, kamrulislamtopu@gmail.com

***Abstract*—The black-box nature of clouds introduces a lack of trusts in clouds. Since provenance can provide a complete history of an entity, trustworthy provenance management for data, application, or workflow can make the cloud more accountable. Current research on cloud provenance mainly focuses on collecting provenance records and trusting the cloud providers in managing the provenance records. However, a dishonest cloud provider can alter the provenance records, as the records are stored within the control of the cloud provider. To solve this problem, we first propose *CloProv* – a provenance model to capture the complete provenance of any type of entities in the cloud. We analyze the threats on the *CloProv* model considering collusion among malicious users and dishonest cloud providers. Based on the threat model, we propose a secure data provenance scheme – *SECProv* for cloud-based, multi-user, shared data storage systems. We integrate *SECProv* with the object storage module of an open source cloud framework – OpenStack Swift and analyze the efficiency of the proposed scheme.**

I. Introduction

The rise of cloud computing has changed the way of using computing services and resources. Today, people are enjoying various services provided by the cloud, such as Dropbox, Office 365, Netflix, Gmail, Google Calendar, and Amazon Elastic Compute Cloud (EC2) instances. While cloud computing is attractive as a cost-efficient and high-performing model, the trustworthiness and accountability of cloud infrastructures have become a rising concern as today's cloud infrastructures often suffer from security issues [1], [2], [3]. Cloud computing appears as a black-box to the end users, which is beneficial to a cloud service provider (CSP) for management and security purposes. Unfortunately, the black-box nature introduces the lack of transparency for cloud providers' activities, which results in distrust and lack of accountability of clouds.

Since provenance information provides the complete history of an entity, i.e., history of the ownership of an entity and the actions performed on that entity [4], incorporating secure provenance as a fundamental property of the cloud can establish trust and accountability of the cloud. Secure provenance tracking can make the cloud compliant with several data protection laws and regulations, such as Sarbanes-Oxley (SOX) [5], Health Insurance Portability and Accountability Act (HIPAA) [6], and Gramm-Leach-Bliley act [7]. Provenance can be also helpful in digital forensics investigation involving clouds for tracking a suspect's activities and establishing a proper chain of custody [8]. By utilizing provenance information, scientists can reason about the origin of data creation, evolution, and flaws in the experiments executed on the cloud.

Past research on cloud provenance mainly focused on modeling, collecting, and querying provenance, leaving security unexplored [9], [10], [11]. State-of-the-art secure provenance schemes [12], [13], [14] cannot be applied in clouds when the CSP is dishonest. The existing works on secure cloud provenance [15], [16] rely on the trustworthiness of CSPs and protect provenance information from external adversaries. However, the honesty of the CSPs cannot be guaranteed. A CSP in its entirety or an employee of the CSP can be dishonest. A cloud is considered as untrusted in contemporary research works [17], [18], [19], [20], [21], [22], [23]. Since all the data and the access histories are under the control of a CSP, a dishonest CSP can always tamper with the provenance records. Moreover, from the provenance data, an attacker can learn confidential information about the data stored in the cloud.

Our Contributions. In this paper, we address the threats of trustworthy provenance management in clouds and propose efficient and secure provenance models while considering that a dishonest CSP can collude with malicious users to alter the provenance information. More specifically, the contributions of this work are the following:

1. We present a novel provenance model – *CloProv* to capture the provenance of any type of entities in clouds. The model complies with the provenance data model (PROV-DM) proposed by W3C [24]. We introduce the notion of *provenance block* and *system provenance* to efficiently manage the provenance records of an entity. The proposed model can motivate researchers to invent secure schemes for various representations of entities.

2. We analyze the threats on *CloProv* considering the CSPs as malicious and collusion between dishonest CSPs and malicious users. Based on this threat model, we then propose a secure data provenance scheme – *SECProv*, which ensures the required security properties. To ensure the integrity of provenance records and chronological ordering of the provenance records, we use the concept of secure provenance chain, introduced in [13] and apply the aggregate signature scheme [25] to efficiently manage the provenance chain. By securely constructing the system provenance using accumulators and publishing the proofs of system provenance to the Internet at the end of every epoch, we can preserve the integrity of the provenance even when the CSP is dishonest.

4. *SECProv* ensures stronger security properties than a state-of-the-art secure provenance scheme – SProv [13]. In SProv, users

have full control over the data and provenance information generated by the users. Whereas, in clouds, the data and the provenance records are under the control of an untrusted CSP. Therefore, SProv cannot guarantee all the security properties required for a cloud-based data storage model when the CSP is dishonest. Managing provenance records using our proposed *CloProv* model and securing the provenance chain using the aggregate signature scheme also make *SECProv* highly storage efficient while incurring very modest performance overhead compared to SProv.

5. We integrate *SECProv* with an open source storage framework for clouds – OpenStack Swift and evaluate the performance of the proposed scheme for various attributes of the system. The overhead for users can be as low as 1.84% and 5.6% to add and update a file respectively, and the average CPU overhead on cloud storage is 2.16%. We also show that *SECProv* is highly storage-efficient compared to SProv.

Outline. Section II introduces the *CloProv* model and presents the threat model on *CloProv* for data provenance. We present the details of *SECProv* scheme in Section III. Section IV states the security analysis of the *SECProv*. Implementation of the proposed model and experimental evaluation are discussed in Section V. Section VI presents the related research works. Section VII concludes and outlines the future research directions.

II. Cloud Provenance Model

A. CloProv Model

In *CloProv*, we use the core structures of PROV-DM [24] and introduce some new attributes to ensure the security of the captured provenance.

• *Entity ($\mathcal{E}$):* An entity ($\mathcal{E}$) is a generic representation of different types of elements, such as files, applications, VMs, etc.

• *Activity (AC):* An activity occurs over a period of time and acts upon or with entities, such as, modifying a file's content.

• *Agent (A):* An agent is responsible for an activity taking place on an entity. Depending on the types of the entity and activity, an agent can be a person or a software/hardware.

• *State (S):* The state of an entity at a given time represents the value of various attributes of the entity. The attributes can vary depending on the types of entities.

• *Provenance Record (PR):* The state of an entity is changed when an agent executes an activity on an entity. A provenance record of an entity represents how the entity comes to a given state from its previous state. According to W3C PROV-DM, a provenance record of an entity should include information about the entity, agent, activity, and the various relations between these three core elements, such as, generation, usage, communication, derivation, attribution, association, delegation, and started time. Since the attributes of states and activities can vary depending on the types of entities, the content of provenance records also varies for different types of entities.

• *Provenance Chain (PC):* The provenance chain of an entity is the sequence of provenance records of the entity ordered by the time. A provenance chain presents how the entity evolves over the time.

• *Provenance Block (PB):* To efficiently manage the provenance chain, we introduce the notion of *provenance block*. A provenance block of an entity holds the provenance records and provenance chain information of a certain epoch. Let us assume that PR^e_0, PR^e_1, $\ldots PR^e_n$ are the provenance records generated for an entity $\mathcal{E}$ in an epoch e. The provenance block of the entity $\mathcal{E}$ for the epoch e, $PB^e_{\mathcal{E}}$, contains the provenance records PR^e_0, PR^e_1, $\ldots PR^e_n$ along with their chronological order information.

• *Provenance Block Chain (PBC):* The provenance block chain of an entity $\mathcal{E}$ is the time-ordered sequence of provenance blocks $PB^0_{\mathcal{E}}|\ PB^1_{\mathcal{E}}|\ \ldots|PB^n_{\mathcal{E}}$ where the sequence of the epochs is $0, 1, \ldots n$.

• *System Provenance (SP):* The system provenance of an epoch contains information about all the provenance blocks generated during the epoch. The system provenance for the epoch e, SP_e, contains information about the provenance blocks $PB^e_{\mathcal{E}_1}$, $\ldots PB^e_{\mathcal{E}_m}$, where $\mathcal{E}_1, \ldots \mathcal{E}_m$ are m different entities.

• *Proof of System Provenance (PSP):* For a system provenance SP_e, there is a proof PSP_e, which is made publicly available to the Internet to ensure the integrity of the system provenance.

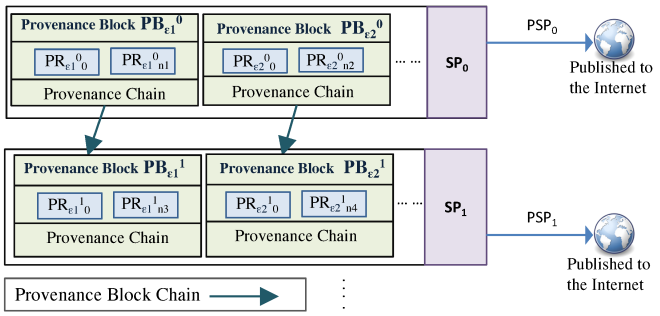


Fig. 1: *CloProv* Model

Figure 1 represents the various components of *CloProv* model. The provenance record generation process along with securing the provenance chain and provenance block chain can vary for different types of entities.

B. Threat Model for Data Provenance

In the data provenance domain, an entity is a document that can be a file or a database tuple stored in the cloud. To design the threat model, we first present the stakeholders below:

• **CSP:** A Cloud Service Provider (CSP) is the owner of a public cloud infrastructure who provides data storage services. The CSP is responsible for generating and storing all the provenance related information. The CSP can be malicious in its entirety or an employee of the CSP can be malicious.

• **Users:** Users are the agents who can access the document hosted in clouds and have read/write access to a document. Multiple users can have read/write access to a single document. A user can be malicious alone or can collude with other users and the CSP.

• **Intruder:** An intruder can be any malicious person including insiders from the CSP who wants to reveal confidential information of a document from the provenance records or to tamper with the provenance information.
• **Auditor:** The role of an auditor is to verify the integrity of the provenance information. The auditor is considered as a trusted entity.

1) Attackers' Capabilities: Unlike the previous works of secure provenance in the cloud [15], [16], we do not consider cloud providers as honest. We also consider that cloud users can be malicious and can collude with the CSP to manipulate the provenance information. Considering the collusion model, attackers are capable to modify provenance records, provenance chains, provenance blocks, and system provenance.

2) Possible Attacks: Below, we present several possible attacks on *CloProv* model in the data provenance domain.

Attacks on Provenance Records and Provenance Chain. Attackers can manipulate information of the provenance records, such as the agent's information, time of an activity, etc. Attackers can also insert a fake provenance record, remove a provenance record, and can reorder the provenance records from a provenance chain. Additionally, all the provenance information of one document can be tagged to another document. The agent of the altered, fake, or deleted provenance record can be the malicious user or other honest users who have access to the document. A malicious user can reorder his/her provenance records, or the provenance records of other honest users.

Repudiation. A dishonest user can repudiate the provenance records where he/she is the agent of the provenance record. A malicious CSP can also repudiate a published proof of system provenance of an epoch.

Attacks on Provenance Blocks and Block Chain. Attackers can remove provenance blocks from the provenance block chain. The order of the provenance blocks in the provenance block chain can also be altered with or without manipulating the provenance chain of the block.

Privacy Violation. A provenance record of a document contains information about the changes to the document, which can leak confidential information for sensitive documents. However, an auditor needs to access the provenance records to verify the integrity of the provenance. Hence, a malicious entity can get access to the confidential provenance information through unauthorized access to the verification procedure. Additionally, since the proofs of system provenance are publicly available on the Internet, an adversary can acquire the published proofs and try to retrieve confidential information from the proofs.

3) System Property: Considering the aforementioned attack scenarios, we argue that a secure data provenance scheme for *CloProv* should ensure the following integrity (I) and confidentiality (C) properties.

I1: One or more malicious users whether acting alone or colluding with the CSP cannot tamper with the provenance records of the malicious user or the provenance records of the other non-colluding honest users.

I2: One or more malicious users whether acting alone or colluding with the CSP cannot plant a fake provenance record in the provenance chain, where the agent in the fake provenance record is one of the malicious user or a non-colluding honest user.

I3: One or more malicious users whether acting alone or colluding with the CSP cannot remove a provenance record from the provenance chain, where the agent of the deleted provenance record is one of the malicious user or a non-colluding honest user.

I4: One or more malicious users whether acting alone or colluding with the CSP cannot reorder the provenance records in the provenance chain, where the agents in those provenance records are the malicious users or other non-colluding honest users

I5: Provenance records of one document cannot be claimed as the provenance records of another document.

I6: A user cannot repudiate provenance chain information. A CSP cannot repudiate any published proof of system provenance.

I7: A malicious CSP cannot tamper with the order of the provenance blocks or cannot remove a block from the provenance block chain.

C1: Adversaries cannot recover any confidential information from the provenance records, provenance chain, and published proofs of system provenance.

III. SECProv: A Secure Data Provenance Scheme for CloProv

A. Building Blocks

Provenance Record. Considering the current state of the document D_i is S_i, a provenance record PR_i for the document D_i can be represented as described in Algorithm 1 using W3C PROV Ontology [26].

Algorithm 1 Representation of a Provenance Record

```
:document_D_i
    a prov:Entity
    prov:wasAttributedTo :U_i
    prov:wasGeneratedBy :documentWriteActivity;
    prov:value Hash(S_i)
:U_i
    a foaf:Person, prov:Agent;
    foaf:openid "Unique ID of U_i";
:documentWriteActivity
    a prov:Activity;
    prov:startedAtTime T_i xsd:dateTime;
    wasAssociatedWith :U_i
:changeInfo
    a prov:Entity
    prov:value Enc_{K_i}(CS_i)
    prov:wasDerivedFrom :document_D_i
:sessionKey
    a prov:Entity
    prov:value Enc_{PK_A}(K_i)
```

In Algorithm 1, $Hash(S_i)$ is a collision-resistance, one-way hash of the current state of the document D_i. U_i is the identity of the actor who is responsible for the change of state of D_i from S_{i-1} to S_i. $Enc_{K_i}(CS_i)$ denotes the encrypted CS_i using the session key K_i, where CS_i describes the changes in the document from the previous state to the current state, such as file difference, the difference in the value of database tuples, etc. T_i is the time when the state of the document is changed from S_{i-1} to S_i. $Enc_{PK_A}(K_i)$ is the encryption of the session key K_i using the public key of the auditor PK_A.

Aggregate Signature. Using the aggregate signature scheme, we can create a secure provenance chain with a single data structure. An aggregate signature scheme is a digital signature that supports aggregation of signatures on a set of distinct messages. Let us assume, there are n users and user U_i has private key SK_i and public key PK_i. A user U_i signs a message M_i to generate a signature σ_i. The aggregate signature algorithm merges all the signatures σ_i $(1 \leq i \leq n)$ and produces a single short signature σ. The aggregation procedure can be also incremental. The verification procedure can decide whether the aggregated signature σ is valid using the public keys of the users' $PK_1, PK_2, \ldots PK_n$, the M distinct message $M_1, M_2, \ldots M_n$, and the aggregated signature σ.

B. Provenance Chain Construction

In the *SECProv* design, the provenance chain for the provenance records of a provenance block is represented by the aggregated signature, σ. To maintain the order of the provenance records while creating the provenance chain, we add the most recent value of hash-chain of the provenance records, HC, with the provenance block. We maintain the chronological ordering of the provenance blocks of a document by adding another item in the provenance block – a hash of the previous block, BC. The complete provenance block for *SECProv* scheme is illustrated in Figure 2.

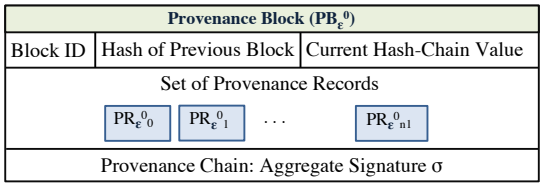


Fig. 2: A Provenance Block for *SECProv*

The chain construction scheme is a tuple of interactive PPT algorithms ConstructChain = (Setup, InitProtocol, PrepProvRecord, GenHashChain, Sign, AppendtoProvChain), such that:

• Setup(g_1, g_2): Let G_1 and G_2 are two (multiplicative) cyclic groups of prime order p which form the bilinear map for the aggregate signature scheme. g_1 is a generator of G_1, g_2 is a generator of G_2. The setup procedure creates a public key and private key for the users. The key generation process works as follows: A user U_j picks a random $x_j \xleftarrow{R} Z_p$, and compute $v_j \leftarrow g_2^{x_j}$. The private key of user U_j is $x_j \epsilon Z_p$ and the public key of the user U_j is $v_j \epsilon G_2$.

• InitProtocol : This algorithm first set up a session key K_i between a user and the CSP using the Diffie-Hellman key exchange protocol. Additionally, if the provenance record is the first record of the provenance block in an epoch, it creates a new provenance block. The provenance block can be the first provenance block for the document. In this case, the value of BC_0 is set to the hash of the first provenance record PR_0 of the block. If there are already one or more provenance blocks for the document, then the value of BC_k for the k^{th} provenance block, PB_k, is generated as: $BC_k \leftarrow Hash(PB_{k-1}), \quad where\ k \geq 1$

While generating a new provenance block, the value of current hash chain, HC, is set to the value of BC.

• $PR_i \leftarrow$ PrepProvRecord($U_j, S_i, CS_i, K_i, PK_A$): When the state of a document is changed from its previous state S_{i-1} to S_i, this algorithm generates the provenance record PR_i according to the Algorithm 1. The confidential information is encrypted using the session key K_i and the session key K_i is encrypted using the public key of the auditor PK_A. We assume that the public-private key pair PK_A, SK_A are known to all the auditors. However, to selectively disclose the records to auditors, we can use broadcast encryption technique [27].

• $HC_i \leftarrow$ GenHashChain (PR_i, HC_{i-1}) : This algorithm generates the hash-chain of the provenance records. For a provenance record PR_i, the hash-chain HC_i is generated as follows:

$$HC_i \leftarrow Hash(PR_i | HC_{i-1}), \quad where\ i \geq 1 \qquad (1)$$

The provenance block stores only the most recent value of the hash-chain.

• $\sigma_i \leftarrow$ Sign(HC_i, x_j): This algorithm creates a signature of the hash-chain value, HC_i, where user U_j is the actor in the provenance record, PR_i. To sign a message, the algorithm first computes $h_i \leftarrow H(HC_i)$, where $h_i \epsilon G_1$, and $\sigma_i \leftarrow h^{x_j}$. The signature $\sigma_i \epsilon G_1$.

• $\sigma \leftarrow$ AppendtoProvChain($\sigma, \sigma_i, HC_i, g_2, v_j$): This algorithm first verifies the signature σ_i on HC_i using the public key v_j of the user U_j. To verify, the algorithm first computes $h_i \leftarrow H(HC_i)$ and accepts the signature σ_i if $e(\sigma_i, g2) = e(h_i, v_j)$ is true.

After the verification of the signature σ_i, the σ_i is appended with the signature σ as follows:

$$\sigma \leftarrow \sigma \times \sigma_i, \ where\ i \geq 1 \qquad (2)$$

For the first provenance record of a provenance block, where i=0, $\sigma \leftarrow \sigma_i$

C. System Provenance Construction

To prepare the system provenance for an epoch, we used an accumulator scheme – Bloom filter, which is used to check whether an item is a member of a set or not [28]. The system provenance construction algorithm works as follows:

• SystemProvenance ← GenSystemProvBloom(Hash of all the provenance blocks of the epoch): The algorithm generates k bit positions for each of the input hash of provenance block by

hashing the item with k different hash functions. The calculated k bit positions of a bit array will be set with 1. The algorithm returns the bit array after inserting the membership information of all the provenance blocks. The bit array is the system provenance SP_e, of an epoch, e.

Proof of System Provenance. We create a proof of the system provenance, PSP_e, for the system provenance, SP_e, as follows;

$$PSP_e =< SP_e, T_p, Sig_{SK_C}(SP_e, T_p) >, \quad (3)$$

Here, T_p is the time of publishing proof, $Sig_{SK_C}(SP_e, T_p)$ is the signature over (SP_e, T_p) using the private key of the CSP, SK_C. The signature is included to ensure the non-repudiation. The proofs will be published to the Internet and can be available by RSS feed to protect it from manipulation by the CSP after publishing the proofs. A blockchain-based technology (commonly associated with modern cryptocurrency [29]) can also be used to ensure the integrity of the proofs since blockchain technology aims to provide a distributed and unalterable ledger of information.

D. Verification

To verify the provenance of a document, an auditor is provided with the provenance blocks generated in the epochs of interest. The provenance records that an auditor wants to verify can be in a single provenance block, in multiple consecutive blocks, or in multiples dispersed blocks.

1) Block Verification: The first step of the overall verification process is block verification. The block verification process first runs the following VerifyProof algorithm:

Result ← VerifyProof(PSP_e,PK_C): Using the public key of the CSP, PK_C, this algorithm verifies the signature of published PSP_e. If the signature is valid, the SP_e and T_p included with the PSP_e are valid and $Result \leftarrow True$; otherwise, $Result \leftarrow False$.

If VerifyProof returns True, the block verification process then runs the following IntegrityBloom algorithm:

• Result ← IntegrityBloom (PB_i,SP_e): The algorithm first generates $HPB_i = Hash(PB_i)$ and then calculates the k bit positions of the Bloom filter by hashing the HPB_i with k different hash functions. These bit positions are then compared with the published SP_e. If all the calculated bit positions are set in the published Bloom filter SP_e, $Result \leftarrow True$; otherwise, $Result \leftarrow False$.

2) Provenance Chain Verification: The provenance chain verification algorithm, VerifyProvChain, verifies the chronological ordering of the provenance records of a provenance block, which works as follows:

• Result ← VerifyProvChain (ProvenanceBlock, HPB_{prev}, σ, g_2):

The algorithm first retrieves n number of provenance records $PR_0, \ldots PR_n$ of the given provenance block. Then it verifies the block chain using BC. If there are no previous provenance blocks for the document, BC should be equal to $Hash(PR_0)$, otherwise, BC will be equal to HPB_{prev}. If the comparison is invalid, $Result \leftarrow False$, otherwise the algorithm continues.

The algorithm then creates the value of hash chain HC_1, HC_2, ... HC_n for the provenance records PR_1, $PR_2, \ldots PR_n$ according to Equation 1 and $HC_0 \leftarrow BC$. Later for all HC_i $(0 \leq i \leq n)$, it computes $h_i \leftarrow H(HC_i)$.

While creating the hash chain values, this algorithm traverses all the provenance records and creates a map between the public key of the user and the h_i values where the user is the actor in PR_i. The set of h_i values for user U_j is denoted as S_j and the i^{th} value of this set is denoted as S_{ji}, where $S_{ji} \epsilon \{h_0, \ldots h_n\}$. Now, if m users are the actors of n provenance records in a provenance block, where $1 \leq m \leq n$, the verification result is computed as follows:

$$Result \leftarrow \begin{cases} True, & e(\sigma, g_2) = \prod_{j=1}^{m} e(v_j, \prod_{i=1}^{nu_j} S_{ji}) \\ False, & Otherwise \end{cases} \quad (4)$$

Here, v_j is the public key of user U_j and nu_j is the number of provenance records, where U_j is the actor of the provenance records.

3) Provenance Block Chain Verification: When the provenance records that need to be verified are in dispersing epochs, an auditor can verify provenance block chain of the intermediate epochs using the following VerifyBlockChain algorithm:

• Result ← VerifyBlockChain (List of provenance blocks, PK_C): This algorithm takes all the provenance blocks of a document which are generated between the start and end epoch. For each block, the algorithm first runs the VerifyProof and then IntegrityBloom algorithm to ensure the integrity of the provenance block. Then for each block, the algorithm checks, whether the BC value of the block is equal to the hash of its previous block. If the chain maintains for all the blocks, we then run the VerifyProvenanceChain algorithm for only those blocks, which holds the provenance records to be verified.

IV. Security Analysis

In this section, we discuss how the *SECProv* scheme ensures the security properties mentioned in the Section II-B3.

I1: If an adversary has altered a provenance record PR_i to PR'_i of a provenance block PB_j, this will change the provenance block from PB_j to PB'_j. Algorithm IntegrityBloom can detect that HPB'_j ($HPB'_j \leftarrow Hash(PB'_j)$) does not exist in the published system provenance.

I2: Let us assume that an adversary wants to add a fake provenance record PR_f between PR_i and PR_{i+1} of the provenance chain, where the adversary is the actor of PR_f. To plant PR_f, the adversary runs following procedures:

• Calculate $HC_f \leftarrow Hash(PR_f|HC_i)$ and $HC'_{i+1} \leftarrow Hash(PR_{i+1}|HC_f)$

• Sign HC_f and HC'_{i+1}

• Since the adversary colludes with the CSP, the CSP can append the signatures on HC_f and HC'_{i+1} with σ.

There can be two different scenarios:

The actor of PR_{i+1} is an honest user U_h: The provenance chain σ holds the signature of U_h on HC_{i+1}. The aggregate

signature scheme supports that the adversary cannot forge the aggregate signature of U_h for message HC'_{i+1} and prepare a σ', where $\sigma' = \sigma$.

The actor of PR_{i+1} is one of the adversaries: In this case, it is possible for an adversary, colluding with the CSP, to build a valid aggregate signature σ' and place it in a provenance block. However, this will change the provenance block from PB_j to PB'_j and the IntegrityBloom algorithm can detect that HPB'_j does not exist in the published system provenance.

I3: Let us assume that PR_{i-1}, PR_i, PR_{i+1} are three consecutive provenance records in the provenance chain and a set of colluding adversaries removed a provenance record PR_i from the provenance chain. To hide the removal of PR_i from the auditor, adversaries run following procedures:

- Calculate $HC'_{i+1} \leftarrow Hash(PR_{i+1}|HC_{i-1})$
- Sign HC'_{i+1}
- Since the adversary colludes with the CSP, the CSP can append the signatures on HC'_{i+1} with σ.

There can be two different scenarios:

The actor of PR_{i+1} is an honest user U_h: While generating the provenance chain, the actual HC_{i+1} was calculated as follows $HC_{i+1} \leftarrow Hash(PR_{i+1}|HC_i)$. Hence, $HC_{i+1} \neq HC'_{i+1}$. Moreover, according to the scheme HC'_{i+1} needs to be signed by a user, U_h. However, it is not possible for an adversary to forge a signature of U_h for message HC'_{i+1}.

The actor of PR_{i+1} is one of the colluding adversaries: In this case, an adversary can provide a valid signature on HC'_{i+1} and CSP replaces σ by σ'. Replacing the actual σ will generate a provenance block that does not exist in the published system provenance. Hence, an auditor can detect such removal.

I4: Let us assume that PR_{i-1}, PR_i, PR_{i+1} are three consecutive provenance records in the provenance chain and the adversaries changed the order to PR_i, PR_{i-1}, PR_{i+1}. To hide the trace of this reordering, the adversaries need to change all the $HC_{i-1}, HC_i, \ and \ HC_{i+1}$ and the signatures $\sigma_{i-1}, \sigma_i, \ and \ \sigma i+1$. If the actors of these three provenance records are honest, adversaries cannot forge the signatures. If all the actors are malicious and collude with the CSP, they can produce $\sigma'_{i-1}, \sigma'_i, \ and \ \sigma i+1'$, which can be aggregated to σ'. However, replacing σ by σ' in the provenance block will produce a provenance block, which does not exist in the system provenance. Hence, such reordering can be detected by an auditor.

I5: The construction of provenance records (Algorithm 1) includes the hash of current state of a document $Hash(S_i)$, which is collision-resistant. Therefore, a provenance record of one document cannot be claimed as the provenance record of another document.

I6: The provenance chain is constructed using the aggregate signature scheme, which requires a user to use the secret key to sign the hash-chain values. Therefore, a user cannot repudiate the provenance chain information. The system provenance is signed by the CSP using the secret key SK_C. Therefore, a CSP cannot deny the published proof of system provenance.

I7: The published proof of system provenance can protect removal or re-ordering of provenance blocks from provenance block chain. The provenance block chain BC is calculated from the hash of the previous block. Removing or reordering of provenance blocks needs altering the value of BC. For example, let us assume PB_0, PB_1, and PB_2 are three consecutive provenance blocks and the adversary changed the order to PB_1, PB_0, and PB_2. To alter the blocks' order, the block chain value of all the provenance blocks need to be changed. In this case, $BC'_1 \leftarrow Hash(First \ PR \ of \ PB_1)$, $BC'_0 \leftarrow Hash(PB_1)$, and $BC_2 \leftarrow Hash(PB_0)$. However, the original values were: $BC_0 \leftarrow Hash \ (First \ PR \ of \ PB_0)$, $BC_1 \leftarrow Hash(PB_0)$, and $BC_2 \leftarrow Hash \ (PB_1)$. Changing the value of BCs will change the provenance blocks and these modified provenance blocks do not exist in the system provenance. Therefore, an auditor can detect any removal or reordering of provenance blocks.

C1: *SECProv* encrypts confidential information with the session key and the session key is encrypted using the public key of the auditor. Therefore, only the auditor can retrieve the session key and will be able to decrypt CS. The provenance chain is generated from the hash-chain values HC, where the HC values are generated using Equation 1. The one-way property of the hash-function and the secrecy of the user's private key ensure that no adversary can extract any confidential information from the provenance chain. Since no confidential information can be leaked from provenance records and provenance chain, an adversary cannot recover any confidential information from the proofs of system provenance.

V. Implementation and Evaluation

A. Implementation

Figure 3 illustrates the system design for integrating *SECProv* with the Swift object storage. All the communication between users and the Swift storage occur through a provenance gateway. A user sends a file upload/delete request to the provenance gateway using REST API provided by the gateway. The gateway is comprised of following five modules:

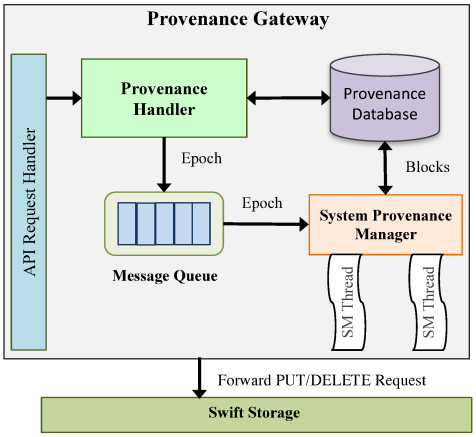


Fig. 3: Integration of *SECProv* with Swift Storage

API Request Handler. The API request handler receives a file upload/delete request from users according to the specified API and sends the request to the provenance handler. The API complies with the existing Swift API.

Provenance Handler. The provenance handler module is responsible for creating the provenance records and the provenance chain according to the scheme presented in Section III-B.

Message Queue. The message queue is used to design asynchronous communication between provenance handler and system provenance manager module. After the end of an epoch, the provenance handler sends the epoch information to the message queue. The message is finally dispatched to a system provenance manager upon request.

System Provenance Manager. After the end of each epoch, the system provenance manager (SM) module creates the system provenance and proofs of system provenance and publishes the proofs to the Internet. All of these procedures are encapsulated in a thread. The system provenance manager always listens for a new message (epoch information) in the message queue. When a new epoch information is found in the message queue, the SM module retrieves the provenance blocks for that epoch and prepares the system provenance and proof of provenance according to the scheme presented in Section III-C. This design provides the flexibility of integrating multiple system provenance managers with a provenance handler.

Provenance Database. The provenance database is managed by a relational database management systems (in our case, it was MySQL) and it stores all the required information to maintain our proposed system: provenance records, provenance blocks, and system provenance.

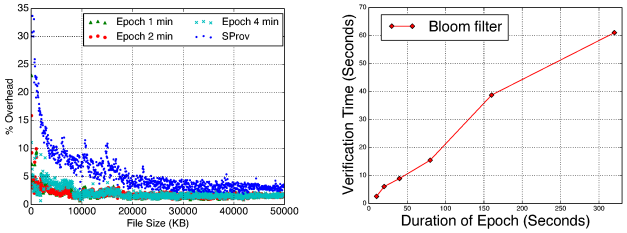
1) System Configuration: We set up the OpenStack Swift storage in an Amazon EC2 [30] m4.large instance (two Intel Xeon processors of 2.40 GHz and 8GB RAM with 30MB cache). We implemented the provenance gateway using JDK 1.8 and MySQL Community Server version 14.14. The API request handler, provenance handler, Swift storage, and the provenance database reside in the same m4.large instance.

We set up the system provenance manager in a separate Amazon EC2 m1.medium instance (one Intel Xeon(R) CPU of 2.40GHz and 3.7GB RAM with 12MB cache). We used Amazon Simple Queuing Service (SQS) [31] for the message queue, which handles the communication between the two instances.

As a client, we used a MacBook Air having 1.7 GHz Intel Core i5 processor and 4 GB RAM. Other than the aggregate signature, we used RSA (2048 bit) for encryption and signature generation and SHA-256 hash functions for hashing. The Bloom filter was constructed with 0.01% false positive probability for 1000 expected number of items.

B. Evaluation

Since SProv [13] is the closest to our work, we compared the performance of *SECProv* with SProv. We used the same security parameters, such as RSA (2048), SHA-256 for implementing SProv.



(a) Overhead for new file upload (b) Provenance verification

Fig. 4: Performance analysis of *SECProv* for various duration of an epoch

Overhead for Uploading a New File. To identify the overhead, we first uploaded 1000 files to the Swift storage without preserving any provenance information and measured the time for each file. The file size was uniformly distributed between 50 KB and 50,000 KB. Then, we uploaded the files to the Swift storage while preserving the provenance information using *SECProv* and measured the time of each file. From the difference of these two calculated times, we identified the percentage overhead with respect to clients for uploading new files. From Figure 4a, we notice that the overhead decreases with the increase in file size and does not vary much with the change of epoch-duration. The reason for such behavior is that the required time for transferring larger files is high, which reduces the overall overhead. The average overhead varies from 1.84% to 2.04% for the different duration of the epoch. As we notice from Figure 4a, the overhead for using *SECProv* is less than the overhead caused by SProv. The overhead for SProv varies from 1.9% to 46.5% and the average overhead is 5.7%.

Overhead for Updating an Existing File. To measure the performance of updating an existing file, we randomly selected a file of size 600 KB and updated the file for 100, 500, 1000, and 2000 times for different epoch-duration. The average overhead for each epoch-duration is approximately 5.6%. However, this overhead is slightly higher compared to the 3.4% average overhead introduce by SProv.

Performance of Provenance Verification. One of the motivations of proposing block-wise provenance management is that the provenance chain verification should perform better when the provenance records are highly dispersed. The experimental results, presented in Figure 4b, justify this assumption. To generate the provenance information, we first updated an existing file 2000 times by varying the duration of the epoch (10, 20, 40, 80, 160, and 320 seconds). To measure the performance of provenance chain verification, we assume that the auditor needs to verify the chronological order of first two provenance records and last two provenance records, which gives us the worst-case scenario of disperse verification. As illustrated in Figure 4b, the verification time increases significantly with the increase in epoch-duration.

With smaller epoch-duration, the number of provenance records in a provenance block gets reduced. Conversely, for longer epoch duration, the number of provenance records in a block gets higher and can eventually turn into one single large block with all the provenance records if the duration

of an epoch is extremely long. Therefore, for short epoch-duration, we can run the VerifyProvenanceChain algorithm for two small, starting and ending provenance blocks and verify the intermediate provenance blocks using VerifyBlockChain algorithm. Therefore, for disperse provenance chain verification, the shorter epoch-duration performs better than longer epoch-duration.

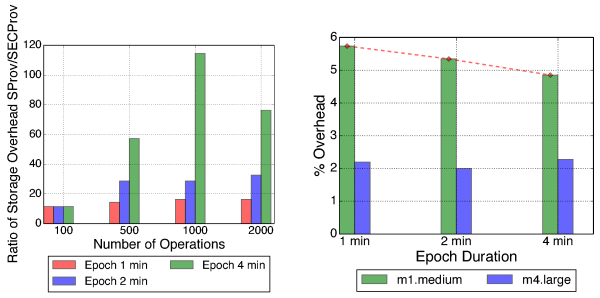
CPU Overhead. We measured the CPU overhead of the two Amazon EC2 instances for different duration of the epoch. The CPU overhead was determined from the CPU performance information of SysBench [32].

First, we measured the CPU performance of the m4.large instance, where the API request handler, provenance handler, provenance database, and the swift storage were running. We first identified the CPU information of the instance when it did not generate any provenance record. Later, we identified the CPU performance of the instance when it served users' requests for updating an existing file with provenance and calculated the overhead. In the same approach, we measured the CPU overhead of the m1.medium instance, where the system provenance manager module was running. We ran the experiment for different duration of epochs to identify the effect of epoch-duration on CPU overhead. The results for CPU overhead are presented in Figure 5b.

As we notice from the Figure 5b, CPU overhead of the m4.large instance does not vary significantly for different epoch duration and the average CPU overhead is 2.16%. However, for m1.medium instance, we notice a downward trend of CPU overhead with the change in epoch duration. It indicates that the CPU overhead of the m1.medium instance (running the system provenance manager module) slowly reduces with the increase in epoch-duration. Increasing the length of epoch-duration reduces the required number of epochs for the same set of files, which also reduces the number of provenance blocks for one file. Therefore, the number of concurrent system provenance manager threads at a certain time gets reduced with the increase in epoch-duration, which justifies the behavior of CPU overhead for the m1.medium instance.

The CPU overhead results also reveal that the system provenance generation and proof publications consume high CPU power. This behavior justifies our design of segregating the system provenance manager module in a separate instance and providing the flexibility of adding multiple system provenance managers as needed.

Storage Overhead. We compared the storage overhead of *SECProv* with the storage overhead of SProv scheme and the results are illustrated in Figure 5a. To measure the storage requirement of *SECProv*, we first identified the number of blocks required to update an existing file for 100, 500, 1000, and 2000 times. For each block, the required storage for *SECProv* is 3004.75 bytes (size of one epoch, one block, and system provenance are 46 bytes, 399 bytes, and 2559.75 bytes respectively). Whereas in SProv, we need to add a signature of 344 bytes (for RSA 2048) with each of the provenance records, which is not needed in *SECProv*. As we notice from Figure 5a, *SECProv* gets more storage-efficient with the increase in epoch-duration for the same number of update operations.



(a) Comparison of storage overhead between *SECProv* and SProv[13]

(b) CPU overhead of the two Amazon EC2 instances

Fig. 5: CPU and storage overhead analysis

VI. Related Works

Hasan *et al.* first introduced secure data provenance [33] and later they proposed a secure data provenance scheme – SProv [13]. They proposed an efficient architecture for capturing provenance information from the application layer. SProv uses an incremental chained signature technique, which can be used by the auditors to verify the integrity. It also ensures that only the authorized auditors can read a provenance record from the chain. The end-to-end provenance system (EEPS) [14] ensures the trustworthiness of the provenance by using trusted provenance monitor. To ensure the trustworthiness of system provenance, Bates *et al.* propose Linux provenance module (LPM) built on top of Linux security modules [12]. These works also do not consider the cloud-base storage while designing the solution. Zhou *et al.* proposed secure network provenance (SNP) [34] for securely constructing network provenance graphs in untrusted environments with Byzantine faults. One of the major assumptions of this work is that the system answer provenance queries observable by at least one correct node. This assumption does not comply with the threat model considered in this paper, when the CSP is dishonest.

Provenance for cloud computing is first proposed by Muniswamy-Reddy *et al.* [9]. The same research group proposed a solution for gathering provenance data from Xen Hypervisor [35]. However, none of the works mentioned above address secure provenance for the cloud where the provenance records are under the control of malicious CSPs. Lu *et al.* first introduced the concept of secure provenance in the cloud [16]. Their scheme provides the provenance of data ownership and process history. The two important stakeholders in the scheme are a trusted system manager (SM) and a trusted CSP. However, a trusted third party, in this case, the SM increases the attack surface and introduces a single point of failure. Moreover, due to the insider attack from a malicious employee, or for incentives, the CSP can also be malicious.

Researchers have proposed various aggregate signature schemes to ensure data integrity [36], [25], [37], [38], [39]. Most of these models depend on user-to-user communication or need TTP. The aggregate signature scheme that we use in

our work is proposed by Boneh *et al.* [25], which does not require user-to-user communication and any TTP. Therefore, we find this scheme best suited to design *SECProv*.

VII. Conclusions and Future Works

Ensuring the trustworthiness of provenance in clouds is challenging because of the possibility of the cloud providers being malicious and colluding with other stakeholders. In this paper, we propose a cloud provenance model *CloProv* to represent the complete provenance of clouds. Using *CloProv*, we propose *SECProv* – a secure provenance scheme for a multi-users, shared, cloud-based data storage system considering the collusion between users and cloud providers. The proposed scheme can ensure the required security properties of trustworthy provenance management in a strong adversarial scenario. We integrated *SECProv* with OpenStack Swift storage and present the efficiency of the scheme. Integrating such a scheme can make clouds more accountable and can attract customers from the business, healthcare, and other sectors that are reluctant in moving towards clouds for trust issue.

In this work, we integrated *SECProv* with OpenStack Swift storage. In future, we plan to incorporate *SECProv* with SQL interfaces and more complex, distributed data management systems, such as DataFlow. The proposed *CloProv* is an abstract model to represent provenance in the cloud, which we use to propose a concrete scheme for a shared data storage model. In the future, we plan to use the *CloProv* model to develop secure provenance schemes for other domains, such as provenance for the state of VMs, and location of data stored in clouds.

Acknowledgements

This research was supported by the National Science Foundation CAREER Award CNS-1351038, ACI-1642078, and DGE-1723768.

References

[1] M. Balduzzi, J. Zaddach, D. Balzarotti, E. Kirda, and S. Loureiro, "A security analysis of amazon's elastic compute cloud service," in *The 27th ACM SAC*, 2012, pp. 1427–1434.

[2] J. Brodkin, "Gartner: Seven cloud-computing security risks," *Infoworld*, pp. 1–3, 2008.

[3] B. R. Kandukuri, V. R. Paturi, and A. Rakshit, "Cloud security issues," in *IEEE SCC'09.*, 2009, pp. 517–520.

[4] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers *et al.*, "The open provenance model core specification (v1. 1)," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 743–756, 2011.

[5] Congress of the United States, "Sarbanes-Oxley Act," http://thomas.loc.gov, 2002, [Accessed July 5th, 2016].

[6] Centers for Medicare and Medicaid Services, "The health insurance portability and accountability act of 1996 (hipaa)," http://www.cms.hhs.gov/hipaa/, 1996, [Accessed July 5th, 2016].

[7] Congress of the United States, "Gramm-leach-bliley financial services mod-ernization act. public law no. 106-102, 113 stat. 1338," 1999.

[8] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *ACM Sigmod Record*, vol. 34, no. 3, pp. 31–36, 2005.

[9] K. Muniswamy-Reddy, P. Macko, and M. Seltzer, "Making a cloud provenance-aware," in *the 1st USENIX TaPP*, 2009.

[10] ——, "Provenance for the cloud," in *The 8th USENIX FAST*, 2010, pp. 15–14.

[11] K. Muniswamy-Reddy and M. Seltzer, "Provenance as first class cloud data," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 11–16, 2010.

[12] A. Bates, D. Tian, K. Butler, and T. Moyer, "Trustworthy whole-system provenance for the linux kernel," in *the 24th USENIX Security Symposium*, 2015.

[13] R. Hasan, R. Sion, and M. Winslett, "The case of the fake Picasso: Preventing history forgery with secure provenance," in *the 7th USENIX FAST*, 2009, pp. 1–12.

[14] P. McDaniel, K. R. Butler, S. E. McLaughlin, R. Sion, E. Zadok, and M. Winslett, "Towards a secure and efficient system for end-to-end provenance." in *TaPP*, 2010.

[15] A. Bates, B. Mood, M. Valafar, and K. Butler, "Towards secure provenance-based access control in cloud environments," in *the 3rd ACM CODASPY*, 2013.

[16] R. Lu, X. Lin, X. Liang, and X. Shen, "Secure provenance: The essential of bread and butter of data forensics in cloud computing," in *the 5th ACM ASIACCS*, 2010, pp. 282–292.

[17] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2386–2396, 2014.

[18] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *the 16th ACM CCS*, 2009, pp. 213–222.

[19] X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, "Outsourcing large matrix inversion computation to a public cloud," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 1–1, 2013.

[20] K. Y. Oktay, M. Gomathisankaran, M. Kantarcioglu, S. Mehrotra, and A. Singhal, "Towards data confidentiality and a vulnerability analysis framework for cloud computing," in *Secure Cloud Computing*. Springer, 2014, pp. 213–238.

[21] Z. Xu, C. Wang, K. Ren, L. Wang, and B. Zhang, "Proof-carrying cloud computation: The case of convex optimization," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1790–1803, 2014.

[22] S. Zawoad, A. K. Dutta, and R. Hasan, "SecLaaS: Secure logging-as-a-service for cloud forensics," in *the 8th ACM ASIACCS*, 2013.

[23] S. Zawoad, R. Hasan, and J. W. Grimes, "Lincs: Towards building a trustworthy litigation hold enabled cloud storage system," *Digital Investigation*, vol. 14, pp. S55–S67, 2015.

[24] "Prov-dm: The prov data model," https://www.w3.org/TR/2013/REC-prov-dm-20130430/, 2013, [Accessed July 14th, 2016].

[25] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Advances in cryptology, EUROCRYPT 2003*. Springer, 2003, pp. 416–432.

[26] "Prov-o: The prov ontology," https://www.w3.org/TR/2013/REC-prov-o-20130430/, 2013, [Accessed July 14th, 2016].

[27] A. Fiat and M. Naor, "Broadcast encryption," in *Advances in Cryptology, CRYPTO93*. Springer, 1994, pp. 480–491.

[28] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[29] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, [Accessed July 14th, 2016].

[30] Amazon EC2, "Amazon elastic compute cloud (amazon ec2)," http://aws.amazon.com/ec2/, [Accessed July 5th, 2016].

[31] Amazon, "Amazon simple queue service (amazon sqs)," http://aws.amazon.com/sqs/, [Accessed August 5th, 2016].

[32] SysBench, "Sysbench: a system performance benchmark," http://sysbench.sourceforge.net/, [Accessed July 14th, 2016].

[33] R. Hasan, R. Sion, and M. Winslett, "Introducing secure provenance: problems and challenges," in *ACM StorageSS '07*, 2007, pp. 13–18.

[34] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr, "Secure network provenance," in *the 23rd ACM SOSP*, 2011, pp. 295–310.

[35] M. Seltzer, P. Macko, and M. Chiarini, "Collecting provenance via the xen hypervisor," in *the 3rd USENIX TaPP*, 2011.

[36] A. Bagherzandi and S. Jarecki, "Identity-based aggregate and multi-signature schemes based on rsa," in *Public Key Cryptography–PKC 2010*. Springer, 2010, pp. 480–498.

[37] C. Gentry and Z. Ramzan, "Identity-based aggregate signatures," in *Public Key Cryptography-PKC 2006*. Springer, 2006, pp. 257–273.

[38] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random oracles," in *Advances in Cryptology-EUROCRYPT 2006*. Springer, 2006, pp. 465–485.

[39] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham, "Sequential aggregate signatures from trapdoor permutations," in *Advances in Cryptology-Eurocrypt 2004*. Springer, 2004, pp. 74–90.