# A Mutual Agreement Signature Scheme for Secure Data Provenance

Mohammed Rangwala*, Zhengli Liang†, Wei Peng*, Xukai Zou* and Feng Li*
*Department of Computer & Information Science, Purdue University, Indianapolis, IN 46202, USA
†Department of Management Science & Engineering, Communication University of China, Beijing 100024, China
Email: {mmrangwa, pengw, fengli}@iupui.edu; liangzhengli@cuc.edu.cn; xkzou@cs.iupui.edu

***Abstract*—Derived from the field of art curation, digital provenance is an unforgeable record of a digital object's chain of successive custody and sequence of operations performed on it. Digital provenance forms an immutable directed acyclic graph (DAG) structure. Recent works in digital provenance have focused on provenance generation, storage and management frameworks in different fields. In this paper, we address two important aspects of digital provenance that have not been investigated thoroughly in existing works: 1) capturing the DAG structure of provenance and 2) supporting dynamic information sharing. We propose a scheme that uses signature-based mutual agreements between successive users to *clearly delineate the transition of responsibility of the document* as it is passed along the chain of users. In addition to preserving the properties of confidentiality, immutability and availability for a digital provenance chain, it supports the representation of DAG structures of provenance. Our scheme supports dynamic information sharing scenarios where the sequence of users who have custody of the document is not predetermined. Security analysis and empirical results indicate that our scheme improves the security of the existing Onion and PKLC provenance schemes with comparable performance.**

***Keywords*—Provenance, cryptography, signatures, integrity, confidentiality, availability**

## I. Introduction

Provenance refers to the origin or earliest known history information of an object. It is an important concept in many fields including art, science and computing where it is used to trace an object to its origin. It is used in workflow management systems and processes in physics, astronomy, biology, chemical sciences, earth sciences for maintaining context information, auditing and data replication [1]. It finds applications for intelligent re-use of experiments, fault detection, protection against illegitimate intellectual property claims, and assessments of data quality [2]. Depending on the application domain, different properties of the object can be tracked such as owner information, purpose of its creation, processes undergone, state of the object or material at each stage, etc. Since provenance maintains information about the present and past of an object, it is suitable to assess its trustworthiness [3].

Digital provenance is the provenance associated with digital objects which can be resources in hardware, software, documents, databases and other entities. It maintains information about the chain of successive custody of the object with different users and the sequence of operations performed on it. It can store functional data such as the process results as well as non-functional data such as the performance of each step. Provenance systems are specially designed application-specific frameworks used to collect, analyze and store all metadata information of an object. They can then be queried to obtain the history information, perform audits and validation checks and detect faults.

Provenance can be represented as a causality graph that connects different objects with edges that describe the process by which the object transformation process took place [4]. This forms an immutable directed acyclic graph (DAG) structure. Although an object keeps changing with operations performed on it, its history information does not and so, provenance is immutable. The DAG structure is justified since an object can be copied to multiple instances (or provided as input to multiple processes) and it can be created from a combination of objects (or from outputs of multiple processes). In a graph, these cases represent a node having multiple children or multiple parents respectively. There is no established standard for representing provenance information, but XML is most popularly used [1]. The existing limited security mechanisms for provenance do not appropriately apply to DAG structures.

Depending on the application domain, provenance can be more or less sensitive than the data object itself. Like other information security subjects, it requires confidentiality, integrity and availability, along with efficient and suitable representation. In this, the security requirements of provenance differ from those of traditional data [4]. Thus, a general scheme for secure provenance is needed, which can be modified depending on the application scenario. We propose a scheme that uses signature-based mutual agreements between successive users to secure the provenance chain. It is an interactive protocol that clearly delineates the transition of responsibility of the document as it is passed along the chain of users. A related provenance scheme was proposed by Hasan et. al. [5]. This scheme is referred to as the *Onion* scheme [6] due to its layered provenance format. Wang et. al. showed that the Onion scheme has certain weaknesses and proposed a linked chain structure of provenance using public keys. This scheme is referred to as the Public-key Linked Chain (*PKLC*) scheme [6]. The PKLC scheme works well for distributed information systems but cannot handle all the properties required in a normal digital system. Our solution extends their work and solves the problems associated with it. The contributions of our paper can be summarized as:

- A signature-based mutual agreement scheme is proposed to form links between provenance records. Our scheme provides better security than the Onion scheme [5], and, is an extension and improvement over the PKLC [6] scheme to provide secure provenance in digital systems


978-1-4799-3572-7/14/$31.00 ©2014 IEEE

other than distributed information networks.
- Our scheme can adequately support the representation of DAG structures of provenance.
- It can also support dynamic information sharing scenarios where the next user to whom the data will be passed is not predetermined. A summary of the advantages of our scheme is provided in Table I.
- An analysis of the security of our scheme is provided to show that it satisfies the security requirements of a provenance scheme.
- Experimental evaluations are provided for the overhead of our scheme. The overhead of our scheme for provenance record creation is a little more than the other schemes but we argue that it can be outweighed by the security provided by our scheme. The results show that it performs better than the Onion scheme for provenance chain verification.

The rest of the paper is organized as follows. We highlight the previous proposed mechanisms in Section II. In Section III, we discuss the entities involved in our scheme and the important properties required for its security. We also discuss the attack model and assumptions of our work. Section IV describes our mutual agreement scheme, along with an example and discusses its properties. We analyze our scheme with respect to the attack model in Section V. Performance evaluation and comparison with existing schemes is provided in Section VI. Section VII talks about future work and concludes the paper.

## II. Related Works

Hasan et. al. [7] were among the first to propose the concept of secure provenance. They defined the properties required from a secure provenance scheme along with a threat model and challenges. Braun et. al. [8], [4] recognized that provenance forms a directed acyclic graph (DAG) structure to which traditional security measures cannot be directly applied. Research has, since, been done to develop conceptual frameworks and models for provenance management [9], [10]; to identify the security requirements of provenance systems [11], [12] and provenance management and data forensics in cloud environments [13], [14].

Kairos [2] is an architecture for securing the data authorship and temporal information in provenance records suited for workflow-based grid computing environments. It uses techniques from public key infrastructure (PKI) such as certificate authorities, digital signatures and time stamping protocols to protect provenance records. Alharbi et. al. [15] proposed a privacy-preserving data provenance scheme to ensure the security of provenance for documents on remote servers. The main focus is to preserve the privacy of the users through the use of hash chains and group signature techniques, and employs the use of a trusted authority and trusted servers.

The Onion Scheme [5], [16] is closely related to our work. It defines a provenance record using multiple encrypted fields and an incremental chained signature mechanism. Each record uses a signature over the previous record's signature which gives the chain an onion-like structure. The Onion scheme has certain weaknesses [6]. First, it cannot protect the outermost layers, i.e. the newest records. An insider attacker can easily extract a prefix of the complete provenance chain and insert a new record. Second, the Onion scheme requires the trusted auditor(s) to maintain user-key relationship which violates the confidentiality of the users. Third, the scheme cannot support the DAG structure of provenance. It is restricted to the scenario of a single document being passed along a chain of users.

The PKLC scheme [6] is based on advancements to the Onion scheme applied to a distributed information network. The record format is similar to that of the Onion scheme, but uses public keys of consecutive users to form weak links between their records. This scheme is suitable for distributed information or wireless sensor networks where the sequence of users/nodes is predetermined, making the weak links enough to preserve the integrity of the chain. The auditors know the path of information flow among users and can thus verify the chain of records. But this cannot be applied directly to dynamic information sharing scenarios where the next user is not known till the document gets passed. Our scheme uses a mutual agreement mechanism and multiple fields for signatures to handle such scenarios.

## III. Preliminaries

This section describes and provides definitions for some fundamental concepts required for our scheme.

### A. Entities involved

A **document** $D$ is a data item such as a file, database tuple or network packet for which provenance is to be generated and maintained. In this paper we use the term document abstractly; its exact form is domain- and application-specific.

**Provenance** of a digital document is an account of all the actions performed on it right from the point of creation. Each access to the document can create a provenance record $Pr$; multiple such records are maintained in order by our scheme as a **provenance chain** $Pr_1|Pr_2|\ldots|Pr_n$. Provenance of a document forms a directed acyclic graph (DAG) structure [4]. We refer to the provenance of the document as a 'chain' in this paper because records are arranged sequentially, but they may not be linearly linked to each other. A provenance record must contain relevant information that helps maintain links between the different records that are part of the chain.

**Users** perform operations on the documents, e.g. create, rename, read, write, delete in the case of a file system.

An **auditor** is an entity who can check all provenance information to verify the lineage of a document. An auditor performs an **auditing** activity, which involves traversing the provenance records in the chain and checking their fields to ensure that the chain has not been tampered with. Different users may trust different auditors with sensitive information, thus, a document has a set of auditors who can access different sensitive fields in the records.

An **adversary** has access to the provenance and wants to alter it in some way for malicious intents but remain undetected. An adversary may be a user who is already part of the provenance chain or an outsider.

In this paper, the symbol $|$ represents concatenation, $e_A()$ represents encryption using the public key $K_A$, $sign_A()$ represents a cryptographic signature with private key $K_A^-$ and $h(D_i)$ represents the cryptographic hash of the document $D_i$. A signature involves the hash of the data, and is short for $sign_i(h(D_i))$.

### B. Properties of the provenance scheme

After discussing the fundamental entities, we discuss the properties that our scheme must provide for the provenance

chain. These are extended from the fundamental properties of general data security.

**Confidentiality:** A provenance record may contain sensitive information regarding the operations performed on the document as well as its ownership history that should not be revealed to unauthorized entities. The sensitivity of these fields is domain- and application-specific. For e.g., in an employee review system, the sequence of managers who have added to the review must not be disclosed to the employee. Thus, the ownership information in such a scenario must be kept confidential. Some general properties that are required are: i) An auditor should be able to verify the complete lineage of the document, without access to the sensitive information in the records. ii) Since different users may trust different auditors, the sensitive information may not be revealed to all auditors.

**Integrity:** Integrity of the provenance is of three types [6]:

1) Immutability (Chain Integrity): The provenance chain once formed should not be modifiable, i.e. the order of the records cannot be changed.
2) Data Integrity: The information in the individual provenance records should not be tampered with.
3) Non-repudiation (Origin Integrity): A user's action in the chain cannot be undone, i.e. the user cannot repudiate his actions.

**Availability:** The scheme must ensure that when the document is passed between users, the provenance chain remains intact and is not modified without being detected in the auditing activity.

**Efficiency:** The provenance generation process and scheme should not add a significant overhead on the application.

### C. Attack Model

Here, we briefly discuss some of the goals of the adversary in a digital provenance scheme similar to discussions in [5] [6]. A detailed analysis of attacks and their prevention in our scheme is discussed in Section V. We consider an adversary with the intentions of,

1) obtaining confidential information from the provenance records about the operations performed on the document
2) obtaining information about the ownership history of the document
3) using fake or stolen key-pairs to make their own provenance records un-verifiable
4) modifying existing records (tampering or changing order of records) or adding forged information to the existing provenance chain
5) selectively removing a certain part of the preceding provenance chain.

Our scheme should be designed such that these goals are either prevented or made detectable to an auditor in the auditing activity.

### D. Assumptions

Before we discuss our scheme in detail, we mention our assumptions:

1) Our scheme considers only the format of the provenance records and chain. It does not focus on the storage and maintenance of the chain. Storage systems such as PASS [9] can be used for this purpose.
2) The provenance generation and storing mechanism is not compromised. Our scheme focuses on securing the provenance from attacks after it has been created and stored securely.
3) The keys used for signatures and encrypting the fields are never compromised or revoked.
4) The document and its provenance are inseparable, i.e. when a document is passed, the complete provenance chain is also passed with it. This must be maintained by the provenance storing mechanism.
5) Our scheme relies on transitive trust defined in [6]. Thus, we assume that consecutive pairs of users do not collude.

## IV. Mutual Agreement Signature Scheme

The provenance chain is composed of a sequence of individual provenance records. Each record has the following structure:

$$Pr_i = <U_i,\ O_i,\ h(D_i),\ ChainInfo_i,\ S_i^*,\ C_i,\ P_i^+,\ N_i^*>$$

$\mathbf{U}_i$ contains identity information about the user $i$ creating this provenance record. This information is specific to an application domain. For a file system provenance record, $U_i$ includes user ID, process ID, ipaddress, port, host, time, and so on.

$\mathbf{O}_i$ gives a representation of the sequence of operations or modifications performed on the document by user $i$. This is also dependent on the application domain. For the file system provenance record, $O_i$ includes a file diff, log of changes or operations, or any other reversible representation [5]. It can also contain subfields for representing the different the operations performed by different processes under the same user [6].

$\mathbf{h(D}_i\mathbf{)}$ is the cryptographic hash of the contents of the document $D_i$ after user $i$ performs all operations.

$\mathbf{ChainInfo}_i$ is a representation of the chain of custody of the document tracked from its origin. It is a sequence of the public keys of all users involved with this document from the owner of the document $U_1$ to the current user $U_i$ represented as $K_{Aud}|K_1|K_2|\ldots|K_i$. $K_{Aud}$ is the public key of the auditor, who must begin the provenance chain.

$\mathbf{S}_i^*$ stores symmetric keys that may be used to encrypt the $U_i$ and $O_i$ fields. We adopt the broadcast encryption scheme of [5] to regulate the access for different auditors. Instead of creating multiple encrypted versions of the sensitive fields for each auditor, user $i$ encrypts them with a unique symmetric key $K_s$, and then stores copies of $K_s$ encrypted with the keys $K_{Aud}^-$ of the respective auditors. The $^*$ indicates there may be zero or more symmetric keys depending on whether encryption is required and the number of auditors user $i$ trusts.

$\mathbf{C}_i$ is the digital signature of the current record $i$ signed by the user $U_i$ with key $K_i^-$, represented as:

$$C_i = sign_i(U_i,\ O_i,\ h(D_i),\ S_i)$$

It ensures the integrity of the current record by essentially providing a checksum of the fields of the record.

$\mathbf{P}_i^+$ is the previous digital signature field which is a representation of the previous provenance record in the chain. The $^+$ indicates there may be more than one previous provenance record from different provenance chains. For the first user $U_1$ in the provenance chain, this field is signed by the auditor with key $K_0$.

$\mathbf{N}_i^*$ is the next digital signature field which is a representation of the subsequent provenance record in the chain. The

$^*$ indicates there may be zero or more subsequent provenance record for a split into different provenance chains.

The $P^+$ and $N^*$ fields are crucial for linking the different provenance records into a chain and for easing the verification process. They form the basis for the mutual agreement scheme.

For record *i*:

$P_i = sign_{i-1}(h(D_{i-1}), ChainInfo_{i-1} | K_i, C_{i-1})$
$N_i = sign_{i+1}(h(D_i), ChainInfo_i | K_{i+1}, C_i)$

For record *i+1*:

$P_{i+1} = sign_i(h(D_i), ChainInfo_i | K_{i+1}, C_i)$
$N_{i+1} = sign_{i+2}(h(D_{i+1}), ChainInfo_{i+1} | K_{i+2}, C_{i+1})$

It can be seen that $N_i$ is signed by user $U_{i+1}$ and $P_{i+1}$ is signed by user $U_i$. Also, the fields $N_i$ and $P_{i+1}$ hold signatures over the same data. This agreement is a form of hand-off mechanism where user $U_i$ passes the document to user $U_{i+1}$ and clearly delineates the transition of responsibility of the document. The previous field for the first user in the provenance chain is signed by the auditor with key $K_{Aud}$ and contains an initialization vector *IV*, known to the auditor, in the hash field. This *IV* is different for each provenance chain. The steps followed by a user for creating a provenance record are described in Algorithm 1. It can be seen that the provenance record is constructed incrementally, the fields are created at each step when the user obtains the document, performs operations and passes it to the next user.

**Algorithm 1** Provenance record creation steps

1: User $i$ creates record $Pr_i$:
2: $Pr_i =< U_i, O_i, h(D_i), ChainInfo_i, S_i^*, C_i, P_i^+, N_i^* >$
3: **if** $i = 1$ **then** ▷ User 1 is the creator of the document
4: $U_1 \leftarrow e_{S_1}$(User 1 Information)
5: $O_1 \leftarrow \phi$ ▷ $U_1$ is the creator of the document
6: $h(D_1) \leftarrow$ Hash of document created $D_1$
7: $ChainInfo_1 \leftarrow K_{Aud} | K_1$
8: $S_1 \leftarrow e_{Aud}(K_{S_1})$ ▷ Multiple for different auditors
9: $C_1 \leftarrow sign_1(U_1, O_1, h(D_1), S_1)$
10: $P_1 \leftarrow sign_{Aud}(IV, K_{Aud} | K_1, C_1)$ ▷ Auditor creates unique *IV* for this document
11: $N_1 \leftarrow \phi$
12: **else** ▷ User $i$ gets the document from user *i-1*
13: $ChainInfo_i \leftarrow ChainInfo_{i-1} | K_i$
14: $P_i \leftarrow sign_{i-1}(h(D_{i-1}), ChainInfo_i, C_{i-1})$
15: $N_{i-1} \leftarrow sign_i(h(D_{i-1}), ChainInfo_i, C_{i-1})$
16: User $i$ modifies document $D_{i-1}$ to $D_i$
17: $U_i \leftarrow e_{S_i}$(User $i$ Information)
18: $O_i \leftarrow e_{S_i}$(Operations performed)
19: $h(D_i) \leftarrow$ Hash of modified document $D_i$
20: $S_i \leftarrow e_{Aud}(K_{S_i})$ ▷ Multiple for different auditors
21: $C_i \leftarrow sign_i(U_i, O_i, h(D_i), S_i)$
22: $N_i \leftarrow \phi$
23: **end if**

### A. Properties

We analyze our scheme and prove that it satisfies the properties discussed in Section III-B.

**Confidentiality:** Some of the information contained in the provenance records may be required to be kept confidential, for e.g., proprietary algorithms, identity of the user, etc. Such fields need to be kept accessible only to the trusted auditor or group of trusted auditors. We achieve this by using the broadcast encryption scheme of [5] to encrypt the *U* and *O* fields using a unique symmetric key generated for the record. The *S* field in the provenance record stores the encrypted versions of the symmetric keys. This scheme reduces the number of encrypted copies of the data to be maintained, by maintaining multiple copies of the key instead.

**Integrity:** Any attacks on the provenance chain, such as tampering, addition or removal of records should be detected by an auditor while performing the auditing activity. In order to facilitate this, our scheme uses hashes and signature fields to hold users accountable for their actions. The current signature *C* protects the fields of the same record providing data integrity. The previous *P* and next *N* signature fields form the mutual agreement between users that links the provenance records of the chain when a document is passed, which provides chain integrity. Since each record contains the signatures of the previous as well as next user, origin integrity is also provided.

**Availability:** An auditor can perform auditing as per the steps in Algorithm 2. The scheme ensures that the provenance chain remains intact and any modifications or malicious activity can be detected in the auditing process.

Our scheme relies on transitive trust among users. It can prevent collusion between users as long as they are not consecutive in the chain. If consecutive users collude to modify their provenance records, it is possible for the records to have appropriate signatures for the agreement signature fields. This change can go undetected in the auditing process.

A thorough security analysis with respect to the attack model is provided in Section V.

### B. Example

The mutual agreement signature scheme is better explained referring to a scenario as shown in Figure 1. The steps followed are from Algorithm 1.

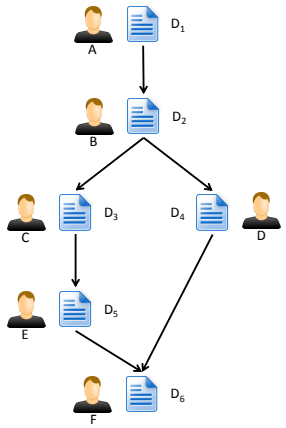



Fig. 1: An example of document passing

The user *A* first creates the document $D_1$. *A* is the owner of this document and its corresponding provenance chain. The provenance record $Pr_A$ generated is:

$Pr_A = <U_A, O_A, h(D_1), ChainInfo_A, S_A^*, C_A, P_A^+, N_A^*>$
$ChainInfo_A = K_{Aud} | K_A;$
$C_A = sign_A(U_A, O_A, h(D_1), S_A);$
$P_A = sign_{Aud}(IV, K_{Aud} | K_A, C_A)$

The operations field $O_A$ is empty, because *A* is the owner and creator of the document $D_1$. The next field $N_A$ is currently empty, because *A* has not yet passed the document to another user. The field $U_A$ can be encrypted using symmetric key $K_{S_A}$, and $K_{S_A}$ can then be encrypted using $K_{Aud}$ and stored in $S_A$. The previous field $P_A$ is signed by the auditor.

When user *B* passes the document $D_2$ to *C* and to *D*, previous fields $P_C$, $P_D$ and and next field $N_B$ are created:

$P_C$ = $sign_B(h(D_2),K_{Aud}|K_A|K_B|K_C,C_B)$
$P_D$ = $sign_B(h(D_2),K_{Aud}|K_A|K_B|K_D,C_B)$
$N_B$ = $sign_C(h(D_2),K_{Aud}|K_A|K_B|K_C,C_B)$;
$sign_D(h(D_2),K_{Aud}|K_A|K_B|K_D,C_B)$

As can be seen, $N_B$ includes two subfields, to show the two separate provenance chains. The data in the signatures is identical to that of the previous fields $P_C$ and $P_D$ which shows the separate agreements for the two chains.

Consider the case when user *F* obtains documents $D_5$ from user *E* and $D_4$ from user *D*. In this case, the previous field $P_F$ has two subfields to indicate the two separate provenance chains. The next fields of the users *D* and *E*, i.e. $N_D$ and $N_E$ are created as:

$N_D$ = $sign_F(h(D_4),K_{Aud}|K_A|K_B|K_D|K_F,C_D)$
$N_E$ = $sign_F(h(D_5),K_{Aud}|K_A|K_B|K_C|K_E|K_F,C_E)$
$P_F$ = $sign_D(h(D_4),K_{Aud}|K_A|K_B|K_D|K_F,C_D)$;
$sign_E(h(D_5),K_{Aud}|K_A|K_B|K_C|K_E|K_F,C_E)$

Again, it can be seen that the data used in the signatures for the subfields of $P_F$ and the fields $N_D$ and $N_E$ are respectively the same indicating the separate agreements for the two chains. The field $ChainInfo_F$ would be created as:

$ChainInfo_F$ = $K_{Aud}|K_A|K_B||K_B$–$K_C$–$K_E$–$K_F|K_B$–$K_D$–$K_F||$

The sequences of public keys from the two branches is concatenated and the order is indicated by the – and || symbols which would represent operators in the storing mechanism. At each step when the document is passed, the complete provenance chain till that point is passed along with it.

### C. Verification/Auditing

The steps followed by an auditor to verify the integrity of the provenance chain are discussed in Algorithm 2. Auditing takes place in reverse order starting from record $Pr_n$ up till the first record $Pr_1$ to be able to verify DAG structures where multiple individual chains may exist. This is different from the verification in the Onion and PKLC schemes. The auditor must verify the hash of the current document, the current signature of each record and the agreements between pairs of records along the chain.

If the current field verification fails for record $Pr_j$, there can be two cases: i) if the agreement between $Pr_j$ and $Pr_{j-1}$ is intact, the record $Pr_j$ has been tampered with, or, ii) if this agreement is not intact, then the current fields and agreements of previous records must be checked till a record $Pr_k$ is found where the current field verification fails but agreement is intact. In such a case the record has been tampered with. A similar case arises if the agreement verification between records $Pr_j$ and $Pr_{j-1}$ fails.

If there is a verification failure at any stage, the auditor (or set of auditors with appropriate symmetric keys of the records) can decrypt the operations fields $O_i$ of the records to reconstruct the document backwards to its original form.

**Algorithm 2** Provenance chain auditing steps

1: $Auditor \leftarrow Pr_1|Pr_2|\ldots|Pr_n; D_n$ ▷ Auditor receives the provenance chain and document $D_n$
2: $H \leftarrow h(D_n)$ ▷ Auditor constructs hash of $D_n$
3: **Verify:** $Pr_n[h(D_n)] = H$
4: **for all** $Pr_i \in Pr_n|Pr_{n-1}|\ldots|Pr_1$ **do**
5: $W \leftarrow Decrypt_{K_i}(C_i)$ ▷ Verifying the current signature field
6: $X \leftarrow h(U_i, O_i, h(D_i), S_i)$
7: **Verify:** $W = X$
8: **if** $P_i = P_1$ **then** ▷ Previous field of record $Pr_1$ involves the auditor
9: $Y \leftarrow Decrypt_{K_{Aud}}(Pr_i[P_i])$
10: $Z \leftarrow h(IV, K_{Aud}|K_1, C_1)$
11: **Verify:** $Y = Z$
12: **end if**
13: **for all** $Pr'_j \in Pr_i[P_i]$ **do** ▷ For each previous field
14: $Y \leftarrow Decrypt_{K_j}(Pr_i[P_i^j])$ ▷ Decrypting previous signature field of $Pr_i$ corresponding to $Pr'_j$
15: $Z \leftarrow Decrypt_{K_i}(Pr'_j[N_j^i])$ ▷ Decrypting next signature field of $Pr'_j$ corresponding to $Pr_i$
16: **Verify:** $Y = Z$
17: **end for**
18: **end for**

With this, each of the verifications can be performed again, to identify the exact instance in the history when the document or its record was tampered with. Any user who can verify the initialization vector used in the previous signature field of the first record can successfully perform auditing of the complete chain. Other users can verify the integrity of the chain beginning from the second record. However, only an auditor can decrypt the symmetric keys stored in the record to further decrypt the *U* and *O* fields.

In our scheme, each record is related only to its previous and next records. Unlike the Onion scheme, where every record contains the checksum information of the complete preceding chain, our scheme involves agreements with only pairs of records. The verification can also be conducted concurrently for pairs of records along the chain similar to the PKLC scheme. If all records are intact, this improves the verification process. If the verification fails for a particular pair of records, the preceding chain can then be investigated following the process previously described.

### D. Advantages of the Mutual Agreement Scheme

The mutual agreement scheme is simple to implement and has many advantages over the schemes discussed in Section II. A summary of these advantages is given in Table I.

First, it provides better protection than the Onion scheme against selective removal of provenance records from the chain. In the Onion scheme, records are loosely linked and do not contain any information about the next or previous user(s). An adversary can selectively remove part $Pr_i|\ldots|Pr_n$ from the chain $Pr_1|Pr_2|\ldots|Pr_n$, append a new record $Pr_i$ with the signature over the signature of record $Pr_{i-1}$ and remain undetected. Our scheme overcomes this drawback by introducing next and previous signature fields which cannot be forged. A record $Pr_i$ contains the signatures of users $U_{i-1}$

| Scheme | Support for DAG structures | Support for dynamic information sharing | Links between records | Link of a record to previous and/or next record |
|---|---|---|---|---|
| Onion Scheme [5] | ✗ | ✓ | Weak because of incremental signatures | ✗ |
| PKLC Scheme [6] | ✓ | ✗ | Only public keys used (weak for dynamic information sharing) | Public keys of previous (optional) and next users |
| Mutual Agreement Scheme | ✓ | ✓ | Strong because of mutual signatures | Signatures of previous and next users on agreements |

TABLE I: Summary of advantages of the Mutual Agreement scheme compared with the Onion and PKLC schemes

and $U_{i+1}$, thus selectively removing a part of the chain as in the case of the Onion scheme will cause the record $Pr_{i-1}$ to contain the signature of the original user $U_i$. Every user has proof of his/her actions, as well as those of the previous user, and information about who the document gets passed to.

Second, our scheme does not require a trusted auditor(s) to know the user-key relationship as in the Onion scheme. This maintains the confidentiality of the users, and also makes the auditor a simpler entity.

Third, our scheme can also support the DAG structure of provenance through the previous $P^+$ and next $N^*$ signature fields. While provenance storage is not our concern in this work, we mention that storing a DAG structured provenance as a linear sequence of records (not linearly linked) would require a topological sorting mechanism. Unlike the PKLC scheme, it can be applied to a dynamic information sharing scenario where the next user is not predetermined. The first record contains the public key of the originator in the previous field of the first record. In a general scenario, this scheme is susceptible to an adversary removing the records and claiming to be the creator of the document. Our scheme overcomes this drawback by having the auditor sign the previous field of the first record. It also involves using an initialization vector *IV* only known to the auditor.

In the Onion scheme and the PKLC scheme, the corresponding current signature field signs over all the fields of the record. In our scheme, however, the current signature field does not sign over *ChainInfo* because *ChainInfo* is protected by the *P* and *N* fields. This reduces the overhead of signing over all fields of the record.

## V. Security Analysis

In this section, we discuss how our scheme prevents the attacks from adversaries discussed in Section III-C.

**Proposition 1.** *An adversary cannot repudiate any records in the provenance chain.*

Each user $U_i$ in the chain must use his/her private key $K_i^-$ to sign the current field $C_i$ of the provenance record $Pr_i$. The signature is on the data fields $U_i$, $O_i$ and $h(D_i)$ which essentially define the actions performed and results obtained by the user, and cannot be forged. The involvement of user $U_i$ in the chain is captured not only in the record $Pr_i$, but also in the previous and next records $Pr_{i-1}$ and $Pr_{i+1}$ due to the mutual agreement mechanism. Thus, after user $U_i$ has created record $Pr_i$, he/she cannot repudiate the record.

**Proposition 2.** *An adversary cannot claim that a valid provenance chain belongs to another document having different contents.*

Each record $Pr_i$ contains a field $h(D_i)$ that stores the cryptographic hash of the document corresponding to that record. If an adversary $U_i$ takes the provenance chain of document $D_1$, attaches it to another document $D_2$ and passes it to user $U_j$, $U_j$ will verify that the $h(D_2) \neq h(D_1)$ from the last record in the chain. Thus, $U_j$ will know that the provenance chain does not belong to $D_2$.

**Proposition 3.** *An adversary cannot alter the fields of records from the preceding chain without being detected.*

Every record $Pr_i$ in the chain contains a field for the current signature $C_i$ which establishes that the signer $U_i$ with private key $K_i^-$ is the creator of that record. This signature cannot be forged by another user. Thus, if an adversary alters the fields of a record $Pr_i$, the signature of the fields $<U_i,O_i,h(D_i)>$ will not match $C_i$ when auditing is performed.

**Proposition 4.** *An adversary cannot add fake records into or remove records (alter the order of records) from the beginning or middle of the chain.*

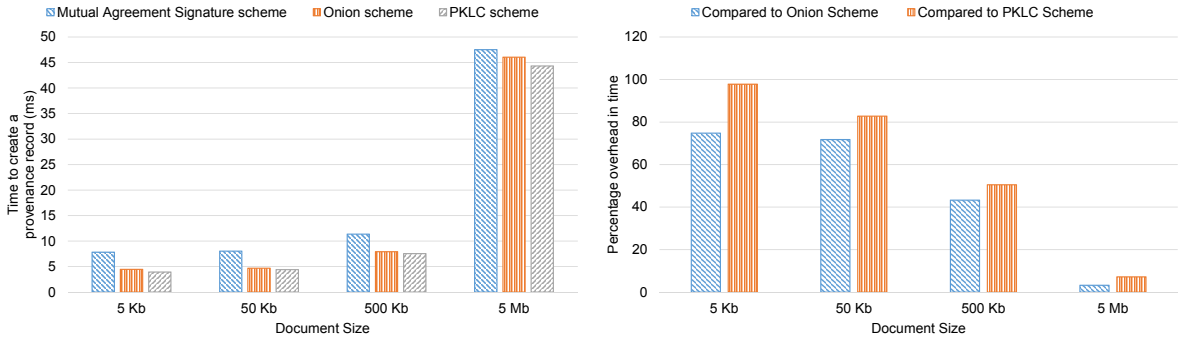
Each record in the provenance chain has previous $P_i$ and next $N_i$ signature fields. These, along with the *ChainInfo* field, denote the order of the users to whom the document was passed. The first record of the owner of the document has the previous field signed by the auditor. Since this signature cannot be forged, an adversary cannot add fake records into the beginning of the chain. If records are added to the middle of the chain, the previous and next fields must by signed by the appropriate users of the chain. Consider the chain contains consecutive records $Pr_i$ and $Pr_j$. If an adversary wants to add a record $Pr_k$ between $Pr_i$ and $Pr_j$, $N_i$ must be signed by user $U_k$ and $P_k$ signed by user $U_i$. Similarly, $N_k$ must be signed by user $U_j$ and $P_j$ signed by user $U_k$. This is not possible since the signatures cannot be forged. Similar arguments hold for the case when records are removed from the beginning or middle of the chain.

**Proposition 5.** *An adversary cannot remove all the records of the chain and claim he/she is the owner of the document by creating a new chain.*

The first record of the owner of the document has the previous field signed by the auditor using the key $K_{Aud}^-$ and contains a unique *IV*. Since this signature cannot be forged, an adversary cannot remove all preceding provenance records and claim to be the owner of the document.

**Proposition 6.** *An adversary cannot modify the document without adding the appropriate provenance record in the chain and not being detected.*

Every provenance record $Pr_i$ stores a reversible representation of the operations $O_i$ performed by the user. If an adversary performs operations on $D_i$ not recorded in $O_i$ or modifies $D_i$ and does not add an appropriate record to the chain, the previous document $D_{i-1}$ cannot be recovered such that its hash is the same as $h(D_{i-1})$ stored in record $Pr_i$. Simple auditing would not require the document to be reconstructed in reverse; however it can be done to resolve




(a) Time to create a provenance record in all three schemes

(b) Percentage overhead of the Mutual Agreement scheme over the Onion and PKLC schemes

Fig. 2: Provenance record construction time in all three schemes and overhead of the Mutual Agreement scheme

discrepancies.

**Proposition 7.** *Two colluding adversaries already having records in the chain cannot add or remove records of other users between their records.*

Consider a provenance chain $Pr_1|\dots|Pr_i|\dots|Pr_j|\dots|Pr_n$. Two colluding adversaries $U_i$ and $U_j$ may want to remove records between their records in the chain, such that the resulting chain is $Pr_1|\dots|Pr_i|Pr_j|\dots|Pr_n$. To be successful, the records $Pr_i$ and $Pr_j$ must have appropriate signatures in the previous and next fields, which is possible in the case of collusion. However, the records must also be updated such that the hashes of the documents at each stage and the operations fields $O_i$ satisfy the reversibility criteria. The case is more difficult when adversaries want to add records of other users between their records because the signatures in all the fields must align which is difficult to achieve.

**Proposition 8.** *An auditor can verify the integrity and availability of the chain but not access any of the confidential information in the records.*

An auditor only requires access to the $<h(D_i),ChainInfo_i,C_i,P_i^+,N_i^*>$ fields of every record in the chain to perform the auditing. User information $U_i$ and operations performed $O_i$ can be kept confidential by encrypting them if required with a session key $K_s$ accessible only to the auditor.

## VI. Implementation & Evaluation

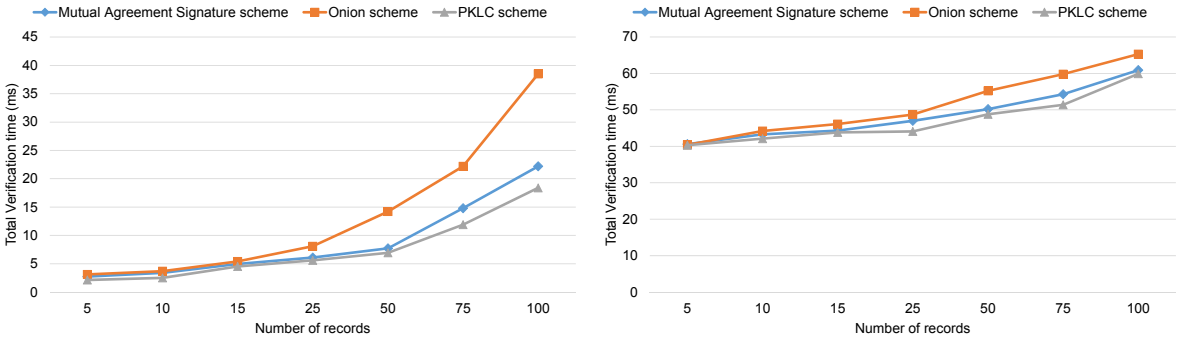
Provenance generation is an overhead in time as well as space that must be considered apart from the security it provides. In this section, we give an evaluation of the overhead of our scheme and compare it with the Onion and PKLC schemes.

We implemented the three schemes, our mutual agreement signature scheme, the Onion scheme and the PKLC scheme with Java JDK 1.7 using NetBeans IDE 7.4 on a Windows 7 machine with Intel Core 2 Duo E6550 2.33GHz processor and 4.0 GB main memory. Since the provenance transmission and storing are not our concern in this work, we simulate the users on the same machine and simulate their activities by modifying the document each time it is passed between users. We use 1024 bit RSA for digital signatures as well as asymmetric encryptions, 128 bit AES for symmetric encryptions, and SHA-1 as our hashing function. We choose four different files 5Kb (text document), 50Kb (text document), 500Kb (JPEG image) and 5Mb (JPEG image) to compare the performance against varying file sizes.

The overhead for creating and verifying the provenance chain can vary depending on the data included in each of the provenance records. We run our experiments under the following settings: i) Each user appends a small amount of information to the file as an operation on it. This ensures that the file size does not change significantly while its hash value does. Since the actual operations performed on the document are not our concern, this suffices for our experiment, ii) Each user trusts the same auditor, and thus the symmetric key used to encrypt the *U* and *O* fields, is encrypted using the public key of only one auditor. iii) For the purpose of the experiments, every user operates on the data only once, however, our scheme allows a user to be repeated in the provenance chain.

**Provenance Record Construction:** The first experiment is to measure the time required by each of the schemes to create a single provenance record. We run this experiment 50 times for each of the varying file sizes and report the average computation time. Figure 2(a) shows the results of the experiment. As expected from the experiments in [6], the PKLC scheme performs better than the Onion scheme. The Onion and PKLC schemes each use two hash operations (one for the file and the other for the current signature field) while our scheme uses four hash operations (two additional for the previous and next fields). As a result, our scheme has a comparatively larger overhead of 97% over the PKLC scheme and 74% over the Onion scheme as seen in Figure 2(b). However, as the file size increases, we observe that the overhead reduces and becomes comparable to the performance of the Onion (7.18% overhead) and PKLC schemes (3.22% overhead). We find that this occurs because as the file size increases, the overhead of the file hash operation alone dominates over the overhead of the other operations. We argue that the security provided by our scheme and support for dynamic information sharing outweighs the overhead of our scheme.

**Provenance Chain Verification:** The next experiment is to measure the time required for an auditor to verify an entire provenance chain. We run this experiment for files of size 50Kb and 5Mb, each for varying number of records in the chain. Each complete chain verification process is run 10 times and we report the average computation time. Since the Onion scheme does not support DAG structures, we experiment with a linear chain to compare their performances. Figure 3 shows the results of auditing the provenance chain for file sizes 50Kb and 5Mb. The verification for the PKLC scheme and our scheme can take place concurrently for pairs of records along the chain. Our scheme takes longer than the




(a) Document size 50Kb

(b) Document size 5Mb

Fig. 3: Provenance chain verification time in all three schemes for file sizes 50Kb and 5Mb

PKLC scheme since it employs two more hashes but performs better than the Onion scheme. As observed in Figure 3(a), for the 50Kb file, the improvement in performance over the Onion scheme becomes significant as the length of the chain increases. The difference grows slower for the 5Mb file since the hash of the file again dominates over the time required by other operations, as seen in Figure 3(b). Since the verification for our scheme and the PKLC scheme happens concurrently over different records, it would be expected that the time required remains fairly constant despite the length of the chain. However, we see an increase in the time, and this we attribute to the overhead of creating and managing threads in Java.

It can be seen from the experiments and our evaluation that our scheme compromises on the performance by providing better security and dynamic information sharing between users.

## VII. Conclusion & Future Work

Provenance is meta-data that stores the history of an object and it has unique security requirements. We proposed a signature-based mutual agreement scheme to protect the confidentiality, integrity and availability of a provenance chain. We discussed the provenance format, verification process and provided an analysis of its security assurance. We gave experimental results of the overhead of our scheme for provenance creation and verification. The overhead for provenance creation is a little more than that of the PKLC and Onion schemes, but it provides better security, whereas our scheme performs better than the Onion scheme for chain verification.

Our scheme depends on transitive trust among users and can detect collusion of users that are not successive. In the future work we will look into a solution for detecting collusion of successive users. Also, our scheme requires a trusted auditor to prevent owner forgery cases and to access confidential information of provenance records. One direction to reduce/remove the involvement of the trusted auditor is to use Hierarchical Access Control (HAC) schemes for efficient key management such that the creator of the document is at the top of the hierarchy. We will also work towards reducing the overhead of our scheme even further, by either employing faster algorithms, or changing the procedure for mutual agreements. In addition, we will implement our scheme in a real world information sharing application and carry out a more thorough security and performance analysis.

## References

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

[2] L. M. R. Gadelha Jr and M. Mattoso, "Kairos: An architecture for securing authorship and temporal information of provenance data in grid-enabled workflow management systems," in *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ser. ESCIENCE '08. IEEE Computer Society, 2008, pp. 597–602.

[3] R. Hasan, "Protecting the Past and Present of Data, with Applications in Provenance and Regulatory-Compliant Databases," *Proceedings of the Third SIGMOD PhD Workshop on Innovative Database Research (IDAR 2009)*, 2009.

[4] U. Braun, A. Shinnar, and M. Seltzer, "Securing provenance," in *Proceedings of the 3rd Conference on Hot Topics in Security*, ser. HOTSEC'08, 2008.

[5] R. Hasan, R. Sion, and M. Winslett, "Preventing history forgery with secure provenance," *ACM Transactions on Storage (TOS)*, vol. 5, no. 4, pp. 12:1–12:43, Dec. 2009.

[6] X. Wang, K. Zeng, K. Govindan, and P. Mohapatra, "Chaining for securing data provenance in distributed information networks," in *Military Communications Conference, 2012 - MILCOM 2012*, Oct 2012, pp. 1–6.

[7] R. Hasan, R. Sion, and M. Winslett, "Introducing secure provenance: Problems and challenges," in *Proceedings of the 2007 ACM Workshop on Storage Security and Survivability*, ser. StorageSS '07, 2007.

[8] U. Braun and A. Shinnar, "A security model for provenance," Computer Science Group, Harvard University, Tech. Rep., 2006.

[9] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems," in *USENIX Annual Technical Conference, General Track*, 2006, pp. 43–56.

[10] S. Sultana and E. Bertino, "A file provenance system," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '13, 2013.

[11] S. Xu, Q. Ni, E. Bertino, and R. Sandhu, "A characterization of the problem of secure provenancemanagement," in *Proceedings of the 2009 IEEE International Conference on Intelligence and Security Informatics*, ser. ISI'09, 2009.

[12] J. Cheney, S. Chong, N. Foster, M. Seltzer, and S. Vansummeren, "Provenance: A future history," in *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA '09, 2009.

[13] K.-K. Muniswamy-Reddy and M. Seltzer, "Provenance as first class cloud data," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 11–16, Jan. 2010.

[14] K.-K. Muniswamy-Reddy, P. Macko, and M. Seltzer, "Provenance for the cloud," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, ser. FAST'10, 2010.

[15] K. Alharbi and X. Lin, "Pdp: A privacy-preserving data provenance scheme," in *Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on*, June 2012, pp. 500–505.

[16] Hasan, Ragib and Sion, Radu and Winslett, Marianne, "The case of the fake picasso: Preventing history forgery with secure provenance," in *Proccedings of the 7th Conference on File and Storage Technologies*, ser. FAST '09, 2009.