IEICE TRANS. INF. & SYST., VOL.E95–D, NO.7 JULY 2012

PAPER

Securing Provenance of Distributed Processes in an Untrusted Environment*

Amril SYALIM[†a)], ***Nonmember***, **Takashi NISHIDE**[†], ***and*** **Kouichi SAKURAI**[†], ***Members***

SUMMARY Recently, there is much concern about the provenance of distributed processes, that is about the documentation of the origin and the processes to produce an object in a distributed system. The provenance has many applications in the forms of medical records, documentation of processes in the computer systems, recording the origin of data in the cloud, and also documentation of human-executed processes. The provenance of distributed processes can be modeled by a directed acyclic graph (DAG) where each node represents an entity, and an edge represents the origin and causal relationship between entities. Without sufficient security mechanisms, the provenance graph suffers from integrity and confidentiality problems, for example changes or deletions of the correct nodes, additions of fake nodes and edges, and unauthorized accesses to the sensitive nodes and edges. In this paper, we propose an integrity mechanism for provenance graph using the digital signature involving three parties: the process executors who are responsible in the nodes' creation, a provenance owner that records the nodes to the provenance store, and a trusted party that we call the Trusted Counter Server (TCS) that records the number of nodes stored by the provenance owner. We show that the mechanism can detect the integrity problem in the provenance graph, namely unauthorized and malicious "authorized" updates even if all the parties, except the TCS, collude to update the provenance. In this scheme, the TCS only needs a very minimal storage (linear with the number of the provenance owners). To protect the confidentiality and for an efficient access control administration, we propose a method to encrypt the provenance graph that allows access by paths and compartments in the provenance graph. We argue that encryption is important as a mechanism to protect the provenance data stored in an untrusted environment. We analyze the security of the integrity mechanism, and perform experiments to measure the performance of both mechanisms.

key words: *provenance security, access control model, database encryption*

1. Introduction

1.1 Background

Provenance of an object is the documentation of the origin and how to produce the object [1]–[6]. It describes the causal relationship between objects, processes and the actors that control the processes and objects. For example, in a hospital, the provenance describes the causal relationships between the objects (medical records, medical test results), the processes that produce the objects (a medical checkup, medical diagnosis) and the actors that control the processes (the physicians). In grid and cloud systems, the provenance records the source of the objects and the processes that affects the condition of objects produced in the system. The provenance is important to verify the quality of the processes and objects.

The provenance can be explicitly recorded and stored along with the objects in the same or different file systems/databases. It can also be later inferred, for example, by asking the actors that control the processes or by checking the computer logs where the processes are executed. Recently, there is much interest in explicit provenance recording where the provenance is recorded in a database (we call the database as a provenance store). Many implementations represent the provenance as a directed graph where the nodes represent the entities (i.e. objects, processes and actors) and edges represent the causal relationship between entities [7], [8].

The provenance has a long history in recording the documentation of valuable objects. In the paper-based world, the provenance is normally recorded as a collection of documents that describe the origin of the object, and all events related to the object that affect the object's current condition. In some fields, for example in works of art, the provenance is regularly used to estimate the quality (and also the value) of the art objects [9]–[11].

With the utilization of computers in almost all aspects of human life, many paper-based provenance systems have been converted to their digital counterparts. Being helped by computers in its recording and storage, the provenance also takes part in improving the quality of data produced by the processes executed by computers [4], [12], [13]. The provenance system has been implemented in computer and grid systems for documenting the processes to produce the data. The provenance system has also been applied in other contexts, for example, in hospitals to document the processes that affect the patient's health condition [14]–[17], and in the courts and police/law institutions [18], [19].

Due to its liquidity, the digital form of provenance is vulnerable to security problems because it can be easily copied, changed, added or deleted by anybody who has access to the provenance store. Ideally, after being recorded in a storage, the provenance should not be altered for any reason. An authorized update to an output of a process (i.e. to correct an error in the output) should be documented in a new provenance record.

Manuscript received August 11, 2011.
Manuscript revised March 3, 2012.

[†]The authors are with the Department of Informatics, Kyushu University, Fukuoka-shi, 819–0395 Japan.

*A preliminary version of this paper has been presented at 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec), June 21-23, 2010.

a) E-mail: amr@itslab.csce.kyushu-u.ac.jp

DOI: 10.1587/transinf.E95.D.1894

1.2 Problem Definition

In a distributed system, the provenance is normally stored in a persistent storage (provenance store, for example a database or a file system) [20] where an interested user can query the provenance to verify the quality of objects produced in the system. Ideally, the storage and the computing environment should be trusted. However, with current technology in computer and network security, it is difficult (if not impossible) to implement a fully trusted storage and computing environment.

There are some integrity and confidentiality mechanisms that can be employed to protect the provenance. The basic integrity mechanism is the digital signature that proves the originality of the provenance. It detects unauthorized updates to the provenance graph (i.e. updates by the person who is not authorized to sign the provenance). It also prevents repudiation by the signer. However, the signature cannot detect malicious updates by an "authorized" person. That is the person who owns the private key for signing the provenance.

This attack ("authorized" update) is viable because normally the parties who are interested in the provenance do not have prior knowledge or copy of the provenance. So, they do not have any evidence about the malicious but "authorized" update that has been made to the provenance. It is not efficient and also costly for each user who is interested in the provenance to make a copy of the provenance promptly after the provenance is submitted to the provenance store. It is also possible that the interested users do not have any previous access to the provenance system, so the users could not make any copies of the provenance.

We show some examples of this attack. The first example is in the process of audit by an external auditor in a company where normally the provenance are kept by the company. When the external auditor inspects the company, without security mechanisms that detect the alteration, the company can re-create a fresh and verifiable provenance. The external auditor cannot detect the alteration because it is signed by authorized parties (if the provenance is created by the people outside the company, they can also collude to alter the provenance). In the context of computer systems, an example is when a user wants to verify the quality of outputs of a grid system in other organizations where the user does not have access previously. Without a secure provenance system, the user does not have a choice other than believing that the provenance is correct and is not altered by "authorized" person in the organization. In the context of the medical record, the medical data of a patient is normally stored in the health care provider of the patient (i.e. the hospital). The hospital can easily change the data without the patient's consent (although the laws in many countries mandate that the data should be under the patient control).

We should also prevent unauthorized access to the sensitive provenance. For example, the medical records contain sensitive data that should not be accessed by unauthorized users. Generally, the access control system to the medical record's database is sufficient to protect the medical records. However, in the case of a higher security requirement, and also whenever the data is stored in an untrusted server (i.e. in the cloud), an alternative method (i.e. encryption) is needed for securing the provenance data.

1.3 Related Work

Two methods to protect integrity of a sequence of digital documents have been proposed by Habert et al. [21]. The first method employs a Trusted Time-Stamping Service (TSS) that issues signed timestamps and also links two timestamps requested consecutively. The TSS links two timestamps by storing the hash value of the first timestamp in the second timestamp. Any changes to the first timestamp can be detected by checking the hash value in the second timestamp. To produce a fake timestamp, the TSS needs to collude with all clients who are requested timestamps after the fake timestamp, so that this scheme assumes the TSS may collude with some clients, but the collusion of the TSS with all clients has a small probability. The second method uses the digital signature to distribute trust among many clients. A client who needs to timestamp a document should ask k random other clients to sign the timestamp. The list of the other clients is generated by a pseudorandom generator that uses hash of the document as a seed. Because the other users are chosen randomly it is assumed that they do not collude to create a fake timestamp. This second method does not employ any TSS but assumes that the users can ask the signatures from the other users.

Hasan et al. [22]–[24] show a threat model for provenance and the method to prevent/detect the attacks associated with the threats by using digital signature, checksum and broadcast/threshold encryption. The provenance is modeled as a chain so the method cannot be applied directly to the graph model. Their method to protect integrity of the provenance chain is by signing each provenance record in the chain and including a checksum of the previous record in the current record to maintain the integrity of the records and the chain structure. They also assume that no collusion of all users (the people who write provenance). For the confidentiality mechanism, Hasan et al. use a broadcast and threshold encryption, and they do not propose a specific access control model.

Gadelha et al. implement a simple time-stamp mechanism for protecting the provenance [25] in a grid system. In their scheme, the provenance is signed by the data owner and hash of the signature is sent to a Time-Stamp Authority (TSA) that appends a timestamp to the hash. The hash and the timestamp is signed by the TSA and send them back as a provenance record receipt. This scheme can prevent repudiation and unauthorized update, but it can not detect a deletion and a malicious "authorized" update to the provenance. Although the process executor cannot update a timestamp, he/she can update the provenance and asks a new timestamp without being detected.

Braun et al. argue that provenance needs a new security model [26]. They also propose a security model for provenance based on observation of the usage of provenance [27]. They focus on the security model but do not deeply discuss how to protect integrity of the provenance. Their main proposal is that we need to control access to heads and tails of the edges and the attributes of the nodes in the provenance graph. However, there is no mechanism proposed to implement their access control model.

1.4 Contributions of This Paper

The contributions of this paper are twofold. First, we propose an integrity mechanism for a directed graph model of the provenance that can solve the problems discussed in Sect. 1.2. We propose a method to record the provenance involving three parties: the process executors that create the provenance, a provenance owner that records the provenance to the provenance store, and a Trusted Counter Server (TCS) that records the number of nodes created by the provenance owner. The process executors need to sign the nodes created by them and include the signatures of inputs they used. The provenance owner needs to send a request to a trusted party (we call the trusted party as the Trusted Counter Server-TCS) to ask a "registration" number before recording a node in the provenance store. The TCS needs to keep the number of nodes stored by the provenance owners. This scheme only assumes trust to the TCS, while the other parties (the process executors and the provenance owner) may cheat.

Our second contribution is introduction of an encryption mechanism for protecting confidential provenance. This mechanism supports an efficient access control to the provenance graph by allowing access based on compartments and access based on the causal relationships (paths in the provenance graph). The key idea is by storing the encryption key of a node in all children of the node. Because when auditing the provenance, a user (i.e. an auditor) normally needs to access the provenance in a path (to check the origin and causal relationships), this method is convenient because we only need to give the key in the leaf node to the user and key to compartments he/she can access.

1.5 Paper Organization

The organization of this paper is as follows. In the next section (Sect. 2) we discuss the provenance system: the provenance model, a uniform DAG representation of provenance, and indirect provenance recording. In Sect. 3, we formalize the notations used in Sect. 4 and Sect. 5. In Sect. 4, we describe the integrity mechanism proposed in this paper. In Sect. 5, we describe the confidentiality mechanism. In Sect. 6, we discuss the experimental results. In the last section (Sect. 7) we conclude the paper.

2. Provenance System

2.1 Modeling Provenance

The provenance of an object captures the information about the process to produce the object [2], [6], [28] that include: (1) the origin/source of the object, and also entities that cause the existence of the object, (2) description about the process to produce the object, and (3) the actor that executes/controls the process. For example, in the medical contexts, the provenance of a medical object (i.e. a medical record) should include the sources of the object (for example, medical tests), the description about the process (i.e. reasoning of diagnosis or the treatment), and also the actor that executes the process (i.e. the physician who write the diagnosis or decides the medical treatments). The provenance can be recorded by the actor that executes the process and can also be recorded by other parties (either manually by people or automatically by computers). On both cases, there should be a proof of the relation between the process with the actor (either by signatures or other proofs).

There are two models that can be used to represent the source/causal relationships between objects. The first model is the chain model that represents the sequential execution of processes that produce the objects [22]–[24]. In this model, the processes are executed one after another where each process uses the output of the process that is executed before the process. The provenance also takes a form of a chain where each link is the documentation of each process. The links are connected for two consecutive processes. Figure 1 shows a provenance of a sequential execution of six processes (*Checkup 1*, *Notes 1*, *Checkup 2*, *Test 1*, *Surgery 1*, and *Result 1*).

The second model supports sequential and also parallel execution of processes. The provenance is modeled by a directed graph where a node represents an entity and an edge represents a causal relationship between two entities (i.e. an object is derived from another object, a process uses an entity as its input) [7], [8]. There should be no cycle in the graph because a node in the provenance graph represents the condition of entity at a specific time [26] (in the case of the same process is repeated, the provenance is recorded in a new node). A relationship between two nodes is a causal relationship, for example a process B uses the output of process A so that the output of A causes the output of B. An annotation can be included in each edge to describe the detail of the causal relationship (i.e. what is the role of an input of a process).

There is no standard of the provenance model although the provenance research community has proposed a provenance model as a standard [2], [29], [30]. The model, that

Fig. 1 Provenance chain of a medical record.

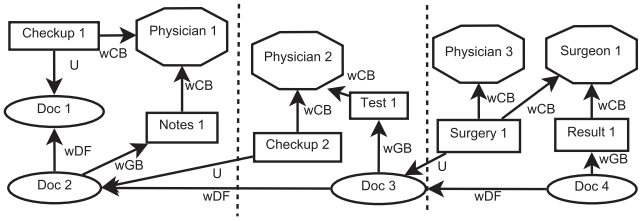

Fig. 2 An example of the Open Provenance Model [5], (U=Used, wDF=wasDerivedFrom, wGB=wasGeneratedBy, wCB=wasControlledBy).

is the Open Provenance Model (OPM), is a directed acyclic graph (DAG) with three types of nodes and five types of edges. The types of nodes are: (1) the artifact, that is immutable information (i.e. an input or an output), (2) the process, that is the series of action on or caused by artifacts and resulting in new artifacts, and (3) the agent is the active entity that starts/controls a process (see Fig. 2). The types of relationships between the nodes are as follows (represented by edges in the graph): (1) an artifact was *used* by a process, (2) an artifact *was generated by* a process, (3) a process *was triggered by* a process, (4) an artifact *was derived from* an artifact, and (5) a process *was controlled by* an agent. The OPM standard does not specify the internal provenance representation and the protocol to store or query the provenance graph to/from a storage [2].

The OPM can be represented graphically by using an octagon as an agent, a rectangle (or a square) to represent a process and an ellipse (or a circle) to represent an artifact. In Fig. 2, we show an example of the OPM in the medical context. This example is adopted from the example of the OPM in [5]. In this example, a patient medical records are created by three physicians and one surgeon (*Physician 1, Physician 2, Physician 3*, and *Surgeon 1*). Initially, the *Physician 1* does a checkup (*Checkup 1*) that uses a previous medical record (*Doc 1*). The *Physician 1* writes a note that is recorded in the *Doc 2*. In the next checkup (*Checkup 2*), the patient meets the *Physician 2* who reads the *Doc 2*, does a test (*Test 1*) that produces *Doc 3*. In the third visit a surgery (*Surgery 1*) is done by *Physician 3* and *Surgeon 1*. They write the result (*Result 1*) in the *Doc 4*.

2.2 A Uniform DAG Representation: Binding the Processes and Artifacts to Agents

In the OPM representation, there are no edges between an agent with an artifact. To know who are responsible for an artifact, we should trace the causal relationship from an artifact to a process and from the process to an agent. In our security mechanism, because an agent should sign the process and its output artifacts, we use a representation of DAG that directly binds each artifact and process. We represent the provenance in a DAG representation (we call a uniform DAG model) where the three types of the entities in the OPM model are represented by a uniform entity: a provenance node. A provenance node represents a computational entity that consists of a process (P), list of references to output artifacts (O), list of references to executing agents (A) and list references to the input artifacts (I). The OPM model can be converted to the uniform representation using Algorithm 1.

Algorithm 1: Converting the OPM model to the uniform DAG model

Input: an OPM graph
Output: the uniform DAG representation
for each OPM node where the type is *process* **do**
  Create a node, where
  Process ← the OPM *process*
  Ouputs ← ref. to artifacts connected with "was generated by"
  Agents ← ref. to agents connected with "was controlled by"
  Inputs ← ref. to artifacts connected with "used", and
  ref. to artifacts connected to *Ouputs* with "was derived by"
end for
for each OPM *artifact* with no "was generated by" connection **do**
  Create a node, where
  Process ← "None"
  Ouputs ← ref. to the OPM *artifact*
  Agents ← ref. to agent of *process* that first uses the OPM *artifact*
  Inputs ← ref. to artifacts connected by "was derived from"
end for
return the uniform DAG nodes

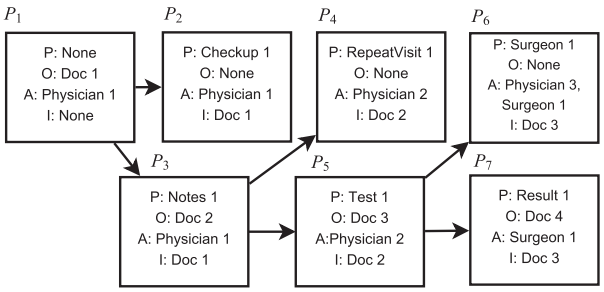

Fig. 3 The uniform DAG model.

Figure 3 shows the result of conversion of the OPM model shown in Fig. 2. In the uniform DAG, in a node only one process documentation is allowed, although there can be many inputs, outputs and agents (an example of a process executed by many agents is the surgery process illustrated in Fig. 2). The uniform DAG representation covers all types of the nodes in the OPM: processes, agents, and input/output artifacts. It also represents four causal relationship between the process, artifacts and agents, because in a node:

1. the outputs (O) "was generated by" the process (P)
2. the process (P) "was controlled by" the agents (A)
3. the inputs (I) are "used" by the process (P)
4. the outputs (O) "was derived from" the input (I)

However, the uniform DAG model does not support a relationship, that is "was triggered by" relationship that represents a relationship between two processes, i.e. process *A* and process *B*, where the process *B* used the output of the process *A* without explicitly defined the output of the process *A*. This relationship exists in an *account* view that represents a less detail process execution where an artifact (that is an output of *A* which is also an input of *B*) is re-

moved from the view. An *account* is useful to simplify the presentation of a provenance graph. The uniform DAG representation does not support *account* because all the outputs and inputs of a process are clearly stated.

2.3 Indirect Provenance Recording

The OPM model does not specify how to record and secure the provenance. A simple model of the provenance recording is by assuming the provenance is created by the process executor and allowing the process executor submits the provenance directly into the provenance store. It is also possible that the provenance is recorded locally by the process executor without sending the provenance to a centralized provenance store. The problem is whenever a party needs to verify the provenance, the party should contact many process executors asking the provenance.

There are three choices of storage of the provenance [3]: (1) no separation of the storage of data and provenance, (2) the data and the provenance are logically separated however in the same physical storage, and (3) the data and the provenance are physically separated. The choice of the storage affects the way to link the provenance and the data. The easiest method of the linking is in the first choice, because we do not need to specify the place of the data and the provenance, they reside in the same storage. In the second and the third storage models, we need to have a linking mechanism that connect data in different storages (logical or physical).

Our scheme does not specify a storage model and the emphasis is in the provenance recording process. In the provenance recording process, the provenance is not stored directly by the process executors, but we introduce another entity, we call the provenance owner who mediates the recording process. A provenance owner can be assigned to be responsible for mediating the provenance recording process in a sub-organization (i.e. a department) or a whole organization.

The mediation by the provenance owner is crucial in our security mechanism, because:

1. The provenance owner has a role as an integrity checker before a provenance node is submitted to the storage. The provenance owner checks the signatures of the provenance node before storing the node to the provenance store.
2. In our integrity scheme, we use a counter system for the integrity checking. The counter is the number of the provenance node stored by the provenance owner and the counter is kept by a trusted party (TCS). The counter should be requested by the provenance owner before storing the provenance node. Besides its security advantage, this method (the provenance mediation) also makes the TCS-system is transparent to the process executors.
3. In our confidentiality scheme, the provenance node is encrypted by the provenance owner because in our security model, the provenance owner is the one that should be responsible for the security of the provenance from unauthorized access (it acts as a security administrator of the provenance).

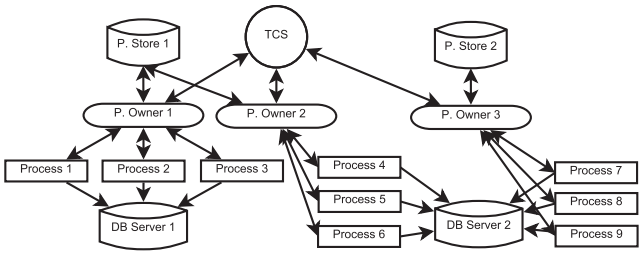

Fig. 4 Our Provenance Recording Architecture.

Our provenance recording architecture is illustrated in Fig. 4. It depicts a design choice of the provenance architecture for distributed processes where the provenance store is separated from the data storage (it should be noted that this storage model is not mandatory, our scheme can use the other storage models). Figure 4 shows a provenance architecture in an organization where there can be many provenance stores (shown in Fig. 4 as P. Stores), many provenance owners that serve many groups of process executors (shown as Processes), and also there can be many data storages referred in Fig. 4 as DB Servers. A Trusted Counter Server (TCS) serves many provenance owners. Its coverage can be in an organization or larger than an organization.

3. Preliminaries

A provenance graph is defined as a set of provenance nodes $P = \{P_0, P_1, \ldots, P_{n-1}\}$ where n is the number of nodes and binary relation E on P that represents the edges such that for $(x, y) \in P \times P$ and $x \neq y$. The relation E maps the provenance nodes in P such that $(x, y) \in P \times P$ and the process in y takes the output in x as its input. We define a tuple as a structure containing multiple elements of the same or different types. A tuple with three elements a, b, c is represented by $\langle a, b, c\rangle$. We also use the notation $\{D_i\}$ to represent the set $D = \{D_0, D_1, \ldots, D_n\}$.

A documentation of a process A is a documentation that describes the steps required to produce the set of outputs D from the set of inputs I. From A we can understand the process to produce the output, what inputs were used, what outputs were produced, how to produce the outputs, when the outputs were produced, and who controlled the process that produced the outputs. The set I contains the inputs that are used by the process. The set D contains the outputs of the process.

The provenance owner PO is an entity that mediates the provenance recording process. A set of process executors C are the actors that controls the execution of a process. A process executor $S \in C$ is a process executor that submits a provenance node to the provenance owner. The Trusted Counter Server TCS is the server that keeps the number

of nodes stored by the provenance owner. The provenance store PS is a database where the provenance is permanently stored. *nid* is a unique identification number for each provenance node. We add a subscript $_{in}$ to a symbol in a provenance node to represent the same symbol in the documentation of processes that produce the inputs. So, that nid_{in} is the provenance node identification number of the process that produce the input. The symbol S_{in} represent a process executor that submit provenance node of an input.

Hash is a cryptographic hash of an object. $Sign$ is a secure digital signature on an object by a subject, for example $Sign_{C_i}(Hash(A, D))$ is a digital signature on hash of A, D by C_i. Enc_{key} is a symmetric encryption function with private key *key*. R is the counter number provided by the TCS for a provenance node. T is a timestamp. We summarize the notations used in this paper in Appendix.

4. Integrity Mechanism for the Provenance Graph

4.1 Provenance Storage

A secure provenance tuple $\mathcal{SP}$ consists of the process documentation A and the integrity data $IntData$ as follows:

$$\begin{aligned}
\mathcal{SP} &= \langle A, IntData\rangle \\
IntData &= \langle IntSub, IntInp\rangle \\
IntSub &= \langle IntProOut, Sign_S(IntProOut)\rangle \\
IntProOut &= \langle Hash(A, D), \{Sign_{C_i}(Hash(A, D))\}\rangle \\
IntInp &= \{\langle IntProOut_{in}, Sign_{S_{in}}(IntProOut_{in})\rangle_i\}
\end{aligned}$$

The $IntData$ consists of $IntSub$ and $IntInp$. To create $IntSub$, each process executor records the hashes of A and all outputs D in a file $Hash(A, D)$ and creates a signature $Sign_{C_i}(Hash(A, D))$ on it. The process executor S combines the hashes $Hash(A, D)$ and all signatures $\{Sign_{C_i}(Hash(A, D))\}$ in a file to form $IntProOut$ and creates signature $Sign_S(IntProOut)$ on it to form $IntSub$. $IntInp$ is the collection of integrity data $IntSub$ of processes that produce the input I.

Each node P_i in the provenance graph is stored in the provenance store in the following format:

$$\begin{aligned}
P_i &= \langle Pid, \mathcal{SP}, T, R, SignData, SID\rangle \\
Pid &= \langle PO, S, nid\rangle \\
SignData &= \langle Sign_S(\mathcal{SP}), Sign_{PO}(RQST), Sign_{TCS}(CNT)\rangle \\
SID &= \{\langle PO, S_{in}, nid_{in}\rangle_i\} \\
RQST &= \langle PO, S, nid, Hash(\mathcal{SP}), T\rangle \\
CNT &= \langle Hash(RQST), R\rangle
\end{aligned}$$

We give an example of the usage of this scheme for P_5 of provenance graph shown in Fig. 3. We use the name of the process, output, agent and input to refer to each process documentation, output, process executor and input. For P_5, the process documentation is A_{Test1}, the input is $Doc2$, the process executor is $Physician2$ and the output is $Doc3$. Because the input is $Doc2$, $IntInp$ is $IntSub$ of P_3 whose process is $Notes1$ and output is $Doc2$. We represent the *nid*, timestamp T and counter R for P_5 as nid_{P5}, T_{P5} and R_{P5} and the *nid* of P_3 is represented by nid_{P3}.

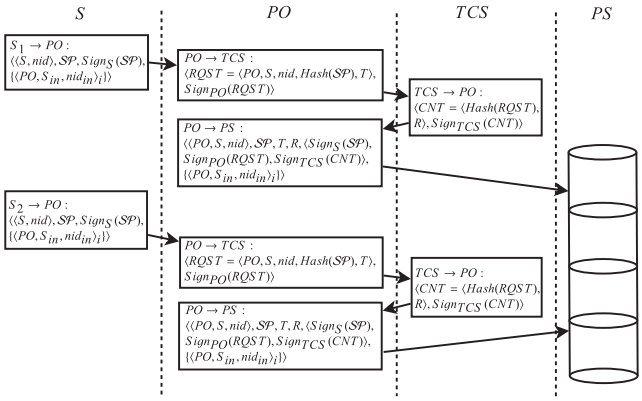

Fig. 5 Secure Provenance Recording Protocol, (S=The Process Executor that submit the provenance node, PO=Provenance Owner, TCS= Trusted Counter Server, PS=Provenance Store).

$$\begin{aligned}
\mathcal{SP} &= \langle A_{Test1}, IntData\rangle \\
IntData &= \langle IntSub, IntInp\rangle \\
IntSub &= \langle IntProOut, Sign_{Physician2}(IntProOut)\rangle \\
IntProOut &= \langle Hash(A_{Test1}, Doc3), \\
&\quad Sign_{Physician2}(Hash(A_{Test1}, Doc3))\rangle \\
IntInp &= \langle IntProOut_{P3}, Sign_{Physician1}(IntProOut)\rangle \\
IntProOut_{P3} &= \langle Hash(A_{Notes1}, Doc2), \\
&\quad Sign_{Physician1}(Hash(A_{Notes1}, Doc2))\rangle
\end{aligned}$$

The provenance node P_5 is stored in the following format:

$$\begin{aligned}
P_5 &= \langle Pid, \mathcal{SP}, T_{P5}, R_{P5}, SignData, SID\rangle \\
Pid &= \langle PO, Physician2, nid_{P5}\rangle \\
SignData &= \langle Sign_{Physician2}(\mathcal{SP}), Sign_{PO}(RQST), \\
&\quad Sign_{TCS}(CNT)\rangle \\
SID &= \langle PO, Physician1, nid_{P3}\rangle \\
RQST &= \langle PO, Physician2, nid_{P5}, Hash(\mathcal{SP}), T_{P5}\rangle \\
CNT &= \langle Hash(RQST), R_{P5}\rangle
\end{aligned}$$

4.2 The Provenance Recording Protocol

The protocol to record provenance consists of three groups of steps as follows (The protocol execution is illustrated in Fig. 5).

1. Creation of the provenance node
2. Requesting the counter from the TCS
3. Storing the provenance node to the provenance store

First: Creation of the provenance node

1. The process receives the inputs, the signatures of the

inputs (the $IntSub$-parts of the documentation of processes that produce the inputs), and the identifications of documentation of the processes that produce the inputs $\{\langle PO, S_{in}, nid_{in}\rangle_i\}$. For a flexible model of execution of the processes, the protocol does not mandate the mechanism to receive the inputs and signatures. The inputs can be received directly from the processes that produce the inputs, queried from the data storage, or by other mechanisms. The signatures can also be received from the processes that produce the inputs, received from the provenance store, or by other mechanisms.

2. The process checks the signatures on the inputs. If the signatures are correct, the process is started.
3. All process executors C create a file that contains hashes of A and all outputs D and signatures by all process executors on the file. The process executor S, who responsible to submit the provenance node creates $IntSub$ that combines $IntProOut$ and signatures $Sign_S(IntProOut)$. The process executor S creates the secure provenance package $\mathcal{SP}$ that consists of a file A and its integrity data $IntData$ as defined in Sect. 4.1.
4. S generates the node identification $\langle S, nid\rangle$, creates a signature on $\mathcal{SP}$, appends a collection of identification of processes that produce the inputs $\{\langle PO, S_{in}, nid_{in}\rangle\}$ and sends $\mathcal{SP}$ along with that information, that is $\langle\langle S, nid\rangle, \mathcal{SP}, Sign_S(\mathcal{SP}), \{\langle PO, S_{in}, nid_{in}\rangle\}\rangle$, to the provenance owner PO.

Second: Requesting the counter from the TCS

1. The provenance owner checks the signature $Sign_S(\mathcal{SP})$, and creates hash of $\mathcal{SP}$, generates a timestamp T, creates $RQST = \langle PO, S, nid, Hash(\mathcal{SP}), T\rangle$, signs $RQST$, and sends $\langle RQST, Sign_{PO}(RQST)\rangle$ to the Trusted Counter Server (TCS).
2. The TCS checks the signature $Sign_{PO}(RQST)$ and the timestamp T and checks how many requests that have been received associated with this provenance owner PO. If the timestamp is within the time X where X is the delay that is acceptable, the TCS increases the number of requests by one, creates the tuple $CNT = \langle Hash(RQST), R\rangle$ and its signature $Sign_{TCS}(CNT)$. The TCS sends CNT and $Sign_{TCS}(CNT)$ to the provenance owner PO.

Third: Storing the provenance node to the provenance store

1. The provenance owner receives CNT and $Sign_{TCS}(CNT)$, and stores the following tuple to the provenance store PS (as defined in Sect. 4.1): $P_i = \langle\langle PO, S, nid\rangle, \mathcal{SP}, T, R, \langle Sign_S(\mathcal{SP}), Sign_{PO}(RQST), Sign_{TCS}(CNT)\rangle, \{\langle PO, S_{in}, nid_{in}\rangle_i\}\rangle$.

4.3 Proof of Correctness of the Integrity Mechanism

Definition 1: Let N be the counter for the provenance owner PO recorded by the TCS (that is the highest R that is issued by TCS for PO). All provenance graphs submitted by PO are consistent if:

1. there is N total number of nodes in the graphs, and
2. for each node in the graphs, all the signatures stored in integrity data $IntData$ and signatures $Sign_C(\mathcal{SP}), Sign_{PO}(RQST)$, $Sign_{TCS}(CNT)$ are correct

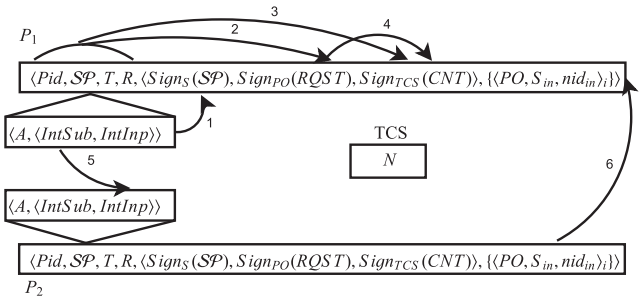

Fig. 6 Integrity Checking.

Theorem 1: By using the above scheme (described in the Sect. 4.2), if the TCS does not collude with any other parties, any other parties cannot make a fake but consistent provenance graph.

Proof 1: We should show that any changes to the provenance graph can be detected. Figure 6 shows the integrity checking for all possible alterations, insertions and deletions done by any parties (except the TCS). It illustrates two nodes connected by an edge (P_2 uses the output of P_1). To update a node P_i consistently, an attacker should also update the signatures $Sign_S(\mathcal{SP}), Sign_{PO}(RQST), Sign_{TCS}(CNT)$, and $IntData = \langle IntSub, IntInp\rangle$. We show that it is not possible to have a consistent provenance graph after any alteration, addition or deletion of the provenance graph:

1. To alter the content of $\mathcal{SP}$ consistently, an attacker should also update the $Sign_S(\mathcal{SP})$ (arrow no 1 in Fig. 6). If the process executor S corrupts and re-creates the signature $Sign_S(\mathcal{SP})$, the alteration can be detected from $Sign_{PO}(RQST)$ (arrow no. 2) because $RQST = \langle PO, S, nid, Hash(\mathcal{SP}), T\rangle$. If the provenance owner PO also corrupts and re-creates the signature $Sign_{PO}(RQST)$, the alteration can be detected from $Sign_{TCS}(CNT)$ (arrows no. 3 and 4) because $CNT = \langle Hash(RQST), R\rangle$. The provenance owner can ask new correct CNT from the TCS, but the TCS will give a new number R so that the total number of nodes will be less than the number recorded by the TCS.
2. To insert a node between two nodes (a parent and a child connected by an edge) consistently, so that the parent will be the parent of the new node and the child will be the child of the new node, an attacker should also update $IntInp$ in the child (arrows 5), because it refers to the $IntSub$ of the previous parent. To alter $IntInp$, an attacker should also alter $Sign_S(\mathcal{SP})$ in the

Table 1 Attack Possibilities and the Integrity Checking.

Attack Possibilities	Integrity Checking
Alteration of a node	checking $Sign_S(\mathcal{SP})$, $Sign_{PO}(RQST)$, and $Sign_{TCS}(CNT)$
Inconsistency of a relationship	checking $IntSub$ and $IntInp$
Deletion of a node	checking R as in corollary 1

child, which is not possible (shown in the proof argument no. 1).

3. To delete a node consistently, deletion causes the number of nodes to decrease, so that the total number of nodes is not the same as N. □

Corollary 1: By using the above scheme (described in the Sect. 4.2), if all provenance graphs created by the provenance owner PO is consistent, for each node whose counter $R \neq 1$ and $R \neq N$, there are two other nodes submitted by PO whose counters are $R+1$ and $R-1$.

Proof 2: Because in a consistent provenance graph, a node cannot be deleted, each node has a unique counter number occupying all numbers from 1 to N where N is the number of nodes created by the provenance owner PO. So that for any number of counter $R \neq 1$ (counter for the first node) and $R \neq N$ (counter the last node) there will be other nodes whose counters are $R+1$ and $R-1$. □

4.4 Checking the Integrity/Consistency of the Provenance Graph

An interested user checks the integrity/consistency of a node or a relationship in the graph using the following method:

1. To check alteration in a node, the user checks signatures $Sign_S(\mathcal{SP})$, $Sign_{PO}(RQST)$, and $Sign_{TCS}(CNT)$. If any of the signatures is not correct, the node has been altered.
2. To check the integrity of relationship between a child and its parents, the user retrieves the parent nodes of the child. For each parent node, the user checks outputs of the parent by comparing $IntSub$ of the parent with $IntInp$ in the child. If the signature is not correct the relationship is not consistent.
3. To find a deletion, for each node, the user checks integrity of the counter R using corollary 1 where for each node whose counter $R \neq 1$ and $R \neq N$, there are two other nodes submitted by PO whose counters are $R+1$ and $R-1$. The user needs to ask the TCS to know N. A node has been deleted if there is a node that does not fulfill the corollary 1.

Table 1 summarizes the integrity checking mechanism. The first column shows the possible attacks, and the second column shows the integrity checking.

4.5 Discussion

4.5.1 Assumptions

To implement this scheme, we need to assume that each process executor, the provenance owner and the TCS have a pair of public key and private key, and each party can retrieve the public keys certificates of the other parties securely. For example in the case of application of provenance in a hospital, each actor (i.e. a physician) should have a pair of public key and the private key. They can also access the public key certificates (to access the public keys) of all other parties securely. We believe this assumption is acceptable because of common usage of the public key system, for example the Public Key Infrastructure (PKI) or alternatively decentralized trust management with the web of trust in PGP [31], [32].

We also assume that a replay attack, where an attacker replays a request that had been sent by the provenance owner, can be detected by using the timestamp T. The timestamp T records the time when the request is made by the provenance owner. The TCS detects the reply attack by recording each request that had been made within a time X and compare each request with at least the requests that had been made within the time X.

4.5.2 The Trusted Counter Server

For the integrity checking, in the TCS we can store information other than the number of nodes, for example the list of the provenance nodes and also hashes/signatures of all nodes. However, this alternative has some drawbacks. The first is we need more storage to store the information because for each node the TCS stores the hashes and signatures. The second drawback is in integrity checking, the user should download all of the integrity data from the TCS while using our method, the user only needs to ask the TCS one time to ask the number N (to check a deletion in integrity checking no. 3). The last drawback is the security mechanism depends only on the TCS while in our model, the security mechanism is distributed among many parties: the process executor, the provenance owner and the TCS.

5. Confidentiality Mechanism for the Provenance Graph

Encryption is an alternative to access control enforced in the provenance store. It is suitable in some situations, for example in the situation where the provenance store cannot be fully trusted (i.e. it resides in a cloud server) or in the situation where the provenance store is highly vulnerable to attackers that break the OS and provenance store access control.

In the encryption method, each sensitive data is encrypted with a key, and all authorized users should be provided with the keys for decrypting the data that he is authorized to access. The provenance store and any attackers breaking the provenance store cannot decrypt the sensitive data without having access to the decryption keys. The problem in the encryption method is how to manage a large number of encryption keys that should be provided to a large number of users with different access policy.

In this section, we show an encryption mechanism for provenance graph that allows path-based access control, so that a user can be granted access to all nodes that have at least a path to a node. This access control is suitable in the provenance graph because a user normally needs to access all nodes that have the causal relationships (connected by paths) with a node. This method is also has a more efficient key management because to grant access to a node and all of its ancestors, we only need to provide a decryption key to access all ancestors. More flexible policies are supported by a compartment-based access control. In compartment-based access control, the access are controlled to groups of nodes (we call the groups as compartments). The compartment-based access control has a higher precedence than the path-based access control, so if a user cannot access a node in a compartment, he/she cannot access the node even if he is allowed to access the node in the path-based access control.

5.1 Provenance Storage

To enforce the path-based access control, the secure provenance tuple $\mathcal{SP}$ is encrypted with node and parent keys. The secure provenance tuple $\mathcal{SP}$ is re-encrypted with the compartment key to enforce the compartment-based access control. We only encrypt the sensitive nodes, so that insensitive nodes that can be accessed by anybody are not encrypted. However, the integrity scheme described in Sect. 4 should be applied to all sensitive and insensitive nodes.

In the encryption scheme, we can enforce both path-based and compartment-based access control, path-based access control only or compartment-based access control only as shown in the schemes as follows:

1. Enforcing both path-based and compartment access control:

$$P_i = \langle Pid, \mathcal{EP}, T, R, SignData, SID, \langle nk, ck\rangle\rangle$$
$$\mathcal{EP} = \langle Enc_{KC}(Enc_{KN}(\mathcal{SP})), Enc_{KK}\langle(KNP, KKP)\rangle\rangle$$

2. Enforcing path-based access control only:

$$P_i = \langle Pid, \mathcal{EP}, T, R, SignData, SID, \langle nk, null\rangle\rangle$$
$$\mathcal{EP} = \langle Enc_{KN}(\mathcal{SP}), Enc_{KK}\langle(KNP, KKP)\rangle\rangle$$

3. Enforcing compartment-based access control only:

$$P_i = \langle Pid, \mathcal{EP}, T, R, SignData, SID, \langle null, ck\rangle\rangle$$
$$\mathcal{EP} = \langle Enc_{KC}(\mathcal{SP}), \langle(KNP, KKP)\rangle\rangle$$

where
– $Pid, \mathcal{SP}, T, R, SignData, SID$ as defined in Sect. 4.1
– $\mathcal{EP}$ = encrypted provenance tuple
– nk = node key generator
– ck = compartment key generator
– KN = the node key
– KC = the compartment key
– KK = the key to encrypt the parent's KN and KC
– KNP = a set of the parent's KC
– KKP = a set of the parent's KK

5.2 Key Generation

The provenance owner store three master keys MKN, MKC, and MKK in a trusted place. Keys KN, KC and KK in each node are generated from MKN, MKC, MKK as follows:

$$KN = Enc_{MKN}(nk)$$
$$KC = Enc_{MKC}(ck)$$
$$KK = Enc_{MKK}(nk)$$

Key generator ck of a node identifies the compartment of the node (the nodes in the same compartments have the same ck) while key generator nk should be unique for each node. When encrypting the node, the provenance owner generates KN, KC, and KK from the master keys MKN, MKC, MKK, nk and ck.

5.3 Encryption Method

The nodes are encrypted by the provenance owner. To encrypt $\mathcal{SP}$ using KN, KC, and KK to produce $\mathcal{EP}$ the provenance owner executes the following steps:

1. The provenance owner generates nk (which is unique for each node) and defines the compartment number ck (where the nodes in the same compartment have the same nk). For path-based access control only, ck is set to *null*. For compartment-based access control only, nk is set to *null*.
2. The provenance owner generates KN, KC, and KK (the mechanism is shown in Sect. 5.2).
3. The provenance owner encrypts the node with the key KN to get $Enc_{KN}(\mathcal{SP})$.
4. The provenance owner re-encrypts the node with key KC to get $Enc_{KC}(Enc_{KN}(\mathcal{SP}))$.
5. The provenance owner creates the tuple $\langle KNP, KKP\rangle$ that is a tuple that contains KN and KK of all parent nodes and encrypts the tuple with key KK to get $Enc_{KK}(\langle KNP, KKP\rangle)$.
6. Encrypted form of the node $\mathcal{EP}$=$\langle Enc_{KC}(Enc_{KN}(\mathcal{SP})), Enc_{KK}(\langle KNP, KKP\rangle)\rangle$.

Figure 7 shows an example of encrypting 4 nodes (P_1, P_2, P_3, and P_4) in the provenance graph. Encrypted forms of the tuples in the nodes are $\mathcal{EP}_1, \mathcal{EP}_2, \mathcal{EP}_3$, and $\mathcal{EP}_4$. There are two compartments: $Comp1$ whose key is KC_1, and $Comp2$ whose key is KC_2. The members of $Comp1$ are P_1 and P_3, the members of the $Comp2$ are P_2 and P_4.

KNP and KKP of a node are all KN and KK in the parent nodes. For example, Fig. 7 shows that KNP and KKP in the node 3 are KN_1, KK_1, KN_2, and KK_2, KN_1, KK_1, KN_2, and KK_2 (which are encrypted with KK_3) because the parent nodes of the node 3 are the node 1 and node 2. The KNP on the $\mathcal{EP}_4$ is KN_3 (the key to encrypt $\mathcal{SP}_3$), and the KKP on the $\mathcal{EP}_4$ is KK_3 (the key to encrypt $\langle KN_1, KK_1, KN_2, KK_2\rangle$.

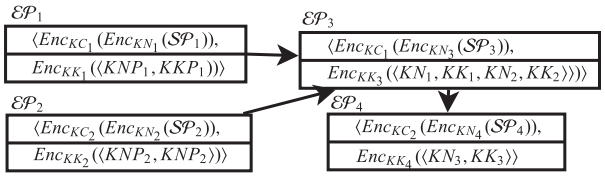

Fig. 7 An example of encryption of 4 nodes in a provenance graph.

5.4 Accessing the Provenance Graph

A user who needs to access the provenance, performs the following steps:

1. The user starts from a leaf node.
2. The user decrypts $Enc_{KC}(Enc_{KN}(\mathcal{SP}))$ with key KC and KN to get $\mathcal{SP}$.
3. The user decrypts $Enc_{KK}(\langle KNP, KKP\rangle)$ by using KK to get KNP and KKP. KNP is a set of KN of the parent nodes and KKP is a set of KK of the parent nodes.
4. By using KNP and KKP, the user decrypts the parent node until reaching a root node. The user can only decrypt the node if he/she has the compartment key KC. However, if the user has KK in a node, he can access all keys of the ancestors of the nodes (all set KNP and KKP).
5. If there is another leaf node, for the new leaf node start from the step 1.

5.5 Access Control Policy

In the naive implementation of access control, the security administrator should define the access policy for each subject and object in the system, which is not efficient and it is difficult to design a consistent policy because there is no structure and common rules in the access policy. Many access control systems improve the efficiency and consistency of access control by supporting an access structure, for example access policy definition based on role, groups or labels [33], [34]. In our access control model for the provenance graph we define structure of access policy by paths and compartments.

By using path-based and compartment-based access control system, the provenance owner defines the policy more efficiently because he/she does not need to define access for each node but he/she can define access to a path and a compartment. The path and compartment structures help the provenance owner to design a consistent policy. Although at first the provenance owner should group the nodes into some compartments, after grouping the nodes, the provenance owner only needs to grant access to a fewer number of compartments rather than a large number of provenance nodes. The provenance owner does not need to define the paths because they are defined from the paths in the provenance graph.

To manage access, first the provenance owner gives access based on the paths. The user who will audit a path should be given access to that path. If the path contains sensitive nodes in some sensitive compartments, the provenance owner decides whether the user be given access to the nodes by including the user to the sensitive compartment. The path-based access control is convenient because a user who audits the provenance normally needs to access all nodes that have a path to the result because the nodes have causal relationship to the result. Compartment-based access control is required to support more general grouping and expressive policies (for example, a user is granted access to a part of the paths).

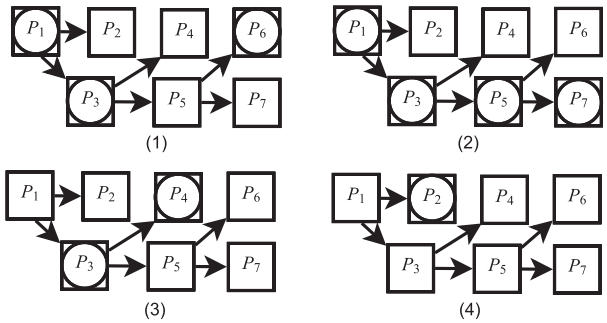

Fig. 8 Sample of access policies.

In Fig. 8, we show an access policy to a provenance graph (that consists of seven nodes) to four users that can be implemented using this encryption mechanism. In this policy, the nodes are divided into two compartments: *confidential* and *unconfidential* where the members of *confidential* are P_4 and P_6. The others are members of *unconfidential*. All users can access the nodes that are members of *unconfidential* so the nodes do not need to encrypted with a compartment key. The first user (no. 1 in Fig. 8) can access P_6 and its ancestors, but he/she cannot access the nodes that are member of *confidential* (P_6 and P_4). The second user (no. 2 in the Figure) can access the node P_7 and its ancestors, and can also access all nodes that are members of *confidential*. The third user (shown in no. 3) can access P_4 and all its ancestors and also nodes in *confidential*. The last user (no. 4) can access P_2, but not its ancestor, and he/she cannot access the nodes that are members of *confidential*. To implement this access policy we provide the four users encryption keys as follows:

- user 1: KN_6 and KK_6
- user 2: KN_7, KK_7, and $KC_{confidential}$
- user 3: KN_4, KK_4, and $KC_{confidential}$
- user 4: KN_2

5.6 Discussion: Key Management

To grant access by providing the encryption keys, as described in Sect. 5.5, first the provenance owner defines access based on the paths. The user who will audit the paths should be provided two keys, they are the node key KN and the parent's key KK in the leaf nodes of the paths. If the path contains sensitive nodes in sensitive compartments, the

provenance owner decides whether the user be given access to the nodes by including the user in the sensitive compartment. To grant access to the compartments, the provenance owner sends the compartment keys to the users. Thus, if the system implements the path-based and compartment-based access control, the number of keys that should be sent to the each user is linear with the number of leaf nodes of the paths and the number of compartments that can be accessed by the user.

Because the number of keys that should be sent to each user is minimal in comparison to the number of nodes, the cost to securely sending the keys (i.e. by using public key encryption) for each user is also minimal. This advantage reduces the computation and network bandwidth that are needed to generate the keys and send the keys to the users. In the basic implementation, to manage the encryption keys, the provenance owner needs to store three master keys MKN, MKC, MKK in a secure storage and generates the keys needed by the users by using the master keys and the key generators nk and ck stored in each node. Alternative implementation is by delegating the key management to a secure key server that stores the master keys and the access policy for each user in the key server. The key servers can compute the keys needed for each users from the master keys and the access policies.

6. Experimental Results

In this section, we describe our experiments to measure the performance of the integrity and confidentiality schemes. To get real data about the performance of the schemes, we develop three applications as follows:

1. The process executor, that is the actor that creates the tuple $\mathcal{SP}$, creates the signature $IntSub$, and sends the tuple and the signature to the provenance owner.
2. The provenance owner, that is the actor that receives the provenance node and the signature, checks the signature, requests the counter to the TCS, encrypts the provenance node and stores the provenance node to the provenance store.
3. The TCS, that is the trusted entity that receives requests from the provenance owner, records the request and reply with the number of requests that have been received from the provenance owner.

6.1 Experimental Setup

The process executor is implemented using Java SE 6 (JCE library for the cryptographic functions: SHA1 for hash and DSA for signature). It sends the provenance node to the provenance owner using HTTP Post protocol. The provenance owner is implemented using PHP and an Apache Web Server. The provenance store is implemented using a Postgresql database. The provenance owner also uses HTTP Post method to send the requests to the TCS. The TCS is implemented using PHP, an Apache web server, and a Postgresql database for storing the counter. The cryptographic functions for digital signature (DSA) in the provenance owner and the TCS are implemented using OpenSSL library while the SHA1 is supported natively by PHP. We use an AES implementation for Windows for encryption [35].

Table 2 Hardware and Software of experiment.

Role	Hardware	Software
Process Executor	Dual-Core 2.50 GHz, 3 GB RAM	Java SE 6 (JCE lib.), Windows XP
Provenance Owner	Dual-Core 2.50 GHz, 3 GB RAM	PHP 5.3.6, OpenSSL lib. ver. 0.9.8, Apache Web Server ver. 2.2.17, Windows XP
TCS	Core 2 Duo 1.4 GHz, 4 GB RAM	PHP 5.3.6, OpenSSL lib. ver. 0.9.8, Apache Web Server ver. 2.2.17, Postgresql ver. 9.0.4, Windows 7
Provenance Store	Core 2 Duo 1.4 GHz, 4 GB RAM	Postgresql 8.4.8, Linux 2.6.32

We performed experiments in four computers where the first computer acts as a process executor that submits the provenance, the second computer acts as the provenance owner, the third computer is the TCS and the fourth computer is the provenance store. All of them are connected by a LAN with speed 100 MB. In Table 2 below, we show the detail specification of the hardware and software we used in the experiments.

We performed 26 experiments to measure the performance of the scheme. For each experiment, we executed the process executor that send the provenance node to the provenance owner. The provenance owner requested the counter and stored the node to the provenance store. For the experiments, we simulated execution of 26 processes that have process documentations A with the range of size from 10 kb to 1237 kb (each process documentation is a text file that describes the execution of a process, the inputs, its outputs and the process executors). Each process use two inputs I and produces two outputs D with size 100 kb each, so that the difference between the processes is only on the size of the process documentation A. In each experiment, we executed the programs 12 times and measured the execution time of various tasks to submit the provenance node. Those tasks are as follows:

1. Hash-sign: the execution time that is needed to create hash and signature of the process documentation by the process executor.
2. Upload: the time to upload the tuple $\mathcal{SP}$ and its signature to the provenance owner
3. Check-PrepReq: the execution time that is needed to check the signature by the provenance owner when receiving the provenance, the time to prepare the request to the TCS (i.e. hash and signature) and the time to receive the response from the TCS (i.e parsing the response and checking the signature of the response)
4. Encrypt1: the execution time that is needed to generate the node key and encrypt with the node key

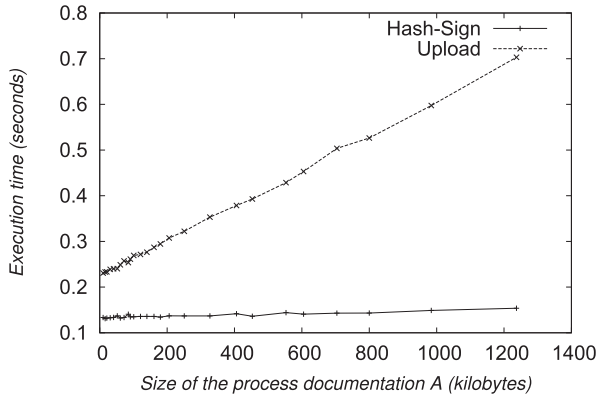

Fig. 9 Execution times of the process executor (in seconds). The list of tasks described in Sect. 6.1 that are shown in this figure are Hash-sign and Upload.

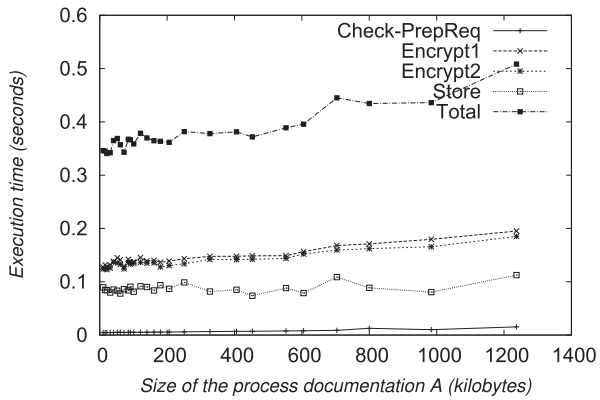

Fig. 10 Execution times of the provenance owner (in seconds). The list of tasks described in Sect. 6.1 that are shown in this figure are Check-PrepReq, Encrypt1, Encrypt2, Store. Total is the total time for those four tasks.

5. Encrypt2: the execution time that is needed to encrypt with the compartment key
6. Req-Counter: the execution time that is needed for sending request until receiving the response from the TCS.
7. Counter: the execution time that is needed to calculate the counter by TCS.
8. Store: the execution time for storing the data to the provenance store.

6.2 Results and Analysis of the Results

Figures 9, 10, and 11 show the average of the execution times of the tasks. The X axis is the size of the process documentation (in kb), the Y axis is the execution time (in seconds) for each process executor, provenance owner and the TCS. We summarize the complexity of the the execution of each task relative to the size of the process documentation *A* in Table 3.

As described in Table 3, for the process executor, the time to create the signature is almost constant. This result

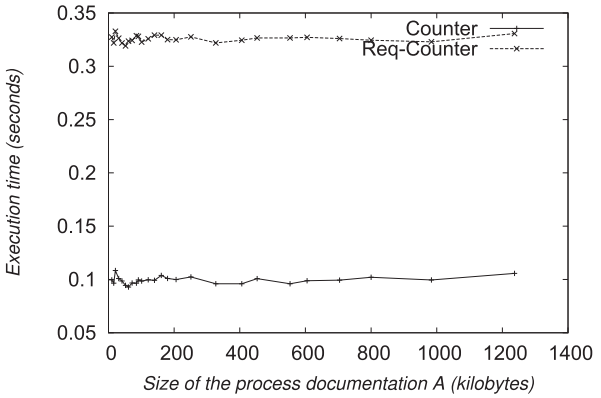

Fig. 11 Execution time of the TCS (in seconds). The list of tasks described in Sect. 6.1 that are shown in this figure are Counter and Req-Counter.

Table 3 The complexity of each task. (relative to the size of process documentation *A*)

Role	Task	Complexity
Process Executor	Hash-Sign	Almost constant
	Upload	Linear
Provenance Owner	Check-PrepReq	Almost Constant
	Encrypt1	Linear (with small growth)
	Encrypt2	Linear (with small growth)
	Store	Almost constant
TCS	Counter	Constant
	ReqCounter	Constant

shows that there is not much difference in the execution time needed to create signature and hash of files in the range of size of the process documentation *A* (10 kb to 1237 kb) and constant size of outputs. The time to upload the provenance node to the provenance owner is linear to the size of the process documentation. This result is natural because the time needed to send the provenance node via the network is linear with the size of the data.

As for the provenance owner, the time to check the signatures and prepare the request is very small and almost constant, while the time to encrypt and store is slightly increased with size. Our analysis is that it is because the algorithms to check the DSA signature, and creating the hash using SHA1 (to prepare the request to the TCS) need almost constant time and the AES encryption algorithm needs time linear with the size of the provenance node. An interesting result is the growth of execution time to submit the provenance node to the provenance store is almost constant, and this growth is different from the growth of time needed by the process executor to upload the provenance node to the provenance owner. These results show that the time needed to upload data using HTTP Post protocol and store the data to a filesystem that are used by the process executor to send the provenance node to the provenance owner is much slower than the protocol to store the data to a Postgresql database using Postgresql library used by the provenance owner to submit the provenance node to the provenance store.

The time that is needed by the TCS to compute the counter and the total time to send request and receive the counter to/from the TCS are also constant. These results are natural because the time to check the signature, to increase counter and prepare the hash is constant. The time to prepare the hash is also constant because the size of the requests is constant (the request consists of the node id, the hash of the provenance node and a timestamp). Because the size of the requests to the TCS and reply (counter) from the TCS are constant, the time needed to send the request and receive the response using the network is also constant.

Our experimental results show the feasibility of implementing our scheme in the real system, because most of the execution times needed in the scheme (except for the time needed to upload the provenance node to the provenance store) are almost constant or with the small growth. Even in our hardware configuration (which is a basic configuration) the time for the TCS to create the counter is around 0.1 seconds and the total time including the network costs (receiving the request and replying with the counter) is not more than 0.5 seconds while for the provenance owner the total time for all tasks except requesting the counter is not more than 0.6 seconds. As for the time to upload the provenance node to the provenance owner our results suggest to use a better or faster protocol and storage than the HTTPS protocol and the normal filesystem.

7. Conclusion

In this paper, we have described the integrity scheme for provenance using digital signature and hash function. We also described the method to protect the confidentiality of the provenance using encryption. Our analysis to the integrity scheme shows that the integrity scheme can detect basic integrity attacks and also integrity attacks namely "authorized" updates, additions and deletions of the provenance. We also shows the advantages of our encryption scheme, that supports for convenient access control policy and key management for managing access to the provenance. Our experimental results show that the integrity and confidentiality scheme is feasible to be implemented in the real systems.

Acknowledgments

We would like to thank Kenichi Takahashi and anonymous reviewers for their useful comments. The first author is supported by the MEXT scholarship. The second and the third authors are partially supported by JAPAN SCIENCE AND TECHNOLOGY AGENCY (JST), Strategic Japanese-Indian Cooperative Programme on Multidisciplinary Research Field, which combines Information and Communications Technology with Other Fields, entitled "Analysis of Cryptographic Algorithms and Evaluation on Enhancing Network Security Based on Mathematical Science."

References

[1] P. Groth and L. Moreau, "Recording process documentation for provenance," IEEE Trans. Parallel Distrib. Syst., vol.20, no.9, pp.1246–1259, 2009.
[2] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P.T. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E.G. Stephan, and J.V. den Bussche, "The open provenance model core specification (v1.1)," Future Generation Comp. Syst., vol.27, no.6, pp.743–756, 2011.
[3] B. Glavic and K.R. Dittrich, "Data provenance: A categorization of existing approaches," Datenbanksysteme in Business, Technologie und Web (BTW 2007), pp.227–241, 2007.
[4] Y. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," SIGMOD Record, pp.31–36, 2005.
[5] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B.M. Thuraisingham, "A language for provenance access control," CODASPY, pp.133–144, 2011.
[6] P. Buneman, S. Khanna, and W.C. Tan, "Why and where: A characterization of data provenance," International Conference on Database Theory (ICDT 2001), LNCS 1973, pp.316–330, 2001.
[7] Y.L. Simmhan, B. Plale, and D. Gannon, "A framework for collecting provenance in data-centric scientific workflows," IEEE International Conference on Web Services (ICWS 2006), pp.427–436, 2006.
[8] S. Bowers, T.M. McPhillips, B. Ludscher, S. Cohen, and S.B. Davidson, "A model for user-oriented data provenance in pipelined scientific workflows," International Provenance and Annotation Workshop (IPAW 2006), LNCS 4145, pp.133–147, 2006.
[9] L. Moreau, P.T. Groth, S. Miles, J. Vázquez-Salceda, J. Ibbotson, S. Jiang, S. Munroe, O.F. Rana, A. Schreiber, V. Tan, and L.Z. Varga, "The provenance of electronic data," Commun. ACM, vol.51, no.4, pp.52–58, 2008.
[10] W.C. Tan, "Research problems in data provenance," IEEE Data Eng. Bull., vol.27, no.4, pp.45–52, 2004.
[11] W.C. Tan, "Provenance in databases: Past, current, and future," IEEE Data Eng. Bull., vol.30, no.4, pp.3–12, 2007.
[12] R.S. Barga and L.A. Digiampietri, "Automatic capture and efficient storage of e-science experiment provenance," Concurrency and Computation: Practice and Experience, vol.20, no.5, pp.419–429, 2008.
[13] S. Miles, P.T. Groth, M. Branco, and L. Moreau, "The requirements of using provenance in e-science experiments," J. Grid Comput., vol.5, no.1, pp.1–25, 2007.
[14] T. Kifor, L.Z. Varga, J. Vázquez-Salceda, S. Álvarez-Napagao, S. Willmott, S. Miles, and L. Moreau, "Provenance in agent-mediated healthcare systems," IEEE Intelligent Systems, vol.21, no.6, pp.38–46, 2006.
[15] V. Deora, A. Contes, O.F. Rana, S. Rajbhandari, I. Wootten, T. Kifor, and L.Z. Varga, "Navigating provenance information for distributed healthcare management," Web Intelligence, pp.859–865, 2006.
[16] S. Álvarez-Napagao, J. Vázquez-Salceda, T. Kifor, L.Z. Varga, and S. Willmott, "Applying provenance in distributed organ transplant management," IPAW, pp.28–36, 2006.
[17] M. Wang, M. Blount, J. Davis, A. Misra, and D.M. Sow, "A time-and-value centric provenance model and architecture for medical event streams," HealthNet, pp.95–100, 2007.
[18] U.M. Maurer, "New approaches to digital evidence," Proc. IEEE, vol.92, no.6, pp.933–947, 2004.
[19] M. Schäler, S. Schulze, and S. Kiltz, "Database-centric chain-of-custody in biometric forensic systems," BIOID, pp.250–261, 2011.
[20] P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsasakou, and L. Moreau, "An architecture for provenance systems," Tech. Rep., University of Southampton, Nov. 2006.
[21] S. Haber and W.S. Stornetta, "How to time-stamp a digital document," J. Cryptology, vol.3, no.2, pp.99–111, 1991.

[22] R. Hasan, R. Sion, and M. Winslett, "Introducing secure provenance: problems and challenges," ACM Workshop on Storage Security and Survivability (StorageSS 2007), pp.13–18, 2007.
[23] R. Hasan, R. Sion, and M. Winslett, "The case of the fake picasso: Preventing history forgery with secure provenance," 7th Conference on File and Storage Technologies (FAST 2009), pp.1–14, 2009.
[24] R. Hasan, R. Sion, and M. Winslett, "Preventing history forgery with secure provenance," ACM Trans. Storage, vol.5, no.4, pp.12:1–12:43, Dec. 2009.
[25] L.M.R. Gadelha and M. Mattoso, "Kairos: An architecture for securing authorship and temporal information of provenance data in grid-enabled workflow management systems," 2008 IEEE Fourth International Conference on eScience, pp.597–602, Dec. 2008.
[26] U. Braun, A. Shinnar, and M. Seltzer, "Securing provenance," The 3rd USENIX Workshop on Hot Topics in Security (HOTSEC 2008), USENIX HotSec, Berkeley, CA, USA, pp.1–5, USENIX Association, July 2008.
[27] U. Braun and A. Shinnar, "A security model for provenance," Tech. Rep., Harvard University, 2006.
[28] S. Miles, P.T. Groth, S. Munroe, S. Jiang, T. Assandri, and L. Moreau, "Extracting causal graphs from an open provenance data model," Concurrency and Computation: Practice and Experience, vol.20, no.5, pp.577–586, 2008.
[29] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson, "The open provenance model," Tech. Rep., University of Southampton, 2007.
[30] L. Moreau, J. Freire, J. Futrelle, R.E. McGrath, J. Myers, and P. Paulson, "The open provenance model: An overview," IPAW, pp.323–326, 2008.
[31] I.T. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A security architecture for computational grids," ACM Conference on Computer and Communications Security, pp.83–92, 1998.
[32] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," IEEE Symposium on Security and Privacy, pp.164–173, 1996.
[33] A. Syalim, Y. Hori, and K. Sakurai, "Grouping provenance information to improve efficiency of access control," International Conference and Workshops on Advances in Information Security and Assurance (ISA 2009), LNCS 5576, pp.51–59, 2009.
[34] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, "Role-based access control models," Computer, vol.29, no.2, pp.38–47, 1996.
[35] AESCrypt, "http://www.aescrypt.com/"

Amril Syalim received B.S. degree from Faculty of Computer Science, the University of Indonesia in 2003, and M.E. degree from Kyushu University in 2006. He is a Lecturer at Faculty of Computer Science, the University of Indonesia since 2007. Since March 2009, he is a PhD. student at Department of Informatics, Kyushu University. His main research is in the areas of database security and applied cryptography.

Takashi Nishide received B.S. degree from the University of Tokyo in 1997, M.S. degree from the University of Southern California in 2003, and Dr.E. degree from the University of Electro-Communications in 2008. From 1997 to 2009, he had worked at Hitachi Software Engineering Co., Ltd. developing security products. Since 2009, he is an assistant professor in Kyushu University. His primary research is in the areas of cryptography and information security.

Appendix: Notations

Table A·1 Notations.

Notation	Explanation
$Hash$	A cryptographic hash of an object
$Sign$	A signature on an object by a subject
Enc_{key}	A symmetric encryption function with private key key
P	A set of the provenance node
A	A documentation about the process execution
D	A set of outputs of a process
I	A set of inputs of a process
C	A set of process executors that responsible to a process
S	A process executor that sends the provenance node
PO	A provenance owner
TCS	The Trusted Counter Server
PS	The Provenance Store
$\mathcal{SP}$	A secure provenance tuple
$IntData$	Integrity data: $\langle IntSub, IntInp\rangle$
$IntInp$	Integrity data for inputs
$IntSub$	Integrity data that consists of $IntProOut$ and signature by S
$IntProOut$	Integrity data for A and outputs D
$IntProOut_{in}$	An $IntProOut$ that referred to the process that produce the input
P_i	A provenance node
Pid	A provenance node identification = $\langle PO, S, nid\rangle$
$SignData$	Signature of S, PO, and TCS
SID	List of inputs' identifications
$RQST$	Request sent to TCS by PO
CNT	Reply by TCS to PO
R	Counter number
T	Timestamp
nid	The node id
S_{in}	A process executor for the process that produce the input and submit its provenance
nid_{in}	The id of the node that produce the input
$\mathcal{EP}$	Encrypted provenance tuple
nk	Node key generator
ck	Compartment key generator
KN	Node key
KC	Compartment Key
KK	Key to encrypt the set of KNP and KKP
KNP	The set of KN of the parent nodes
KKP	The set of KK of the parent nodes
MKN	The master key for generating KN
MKC	The master key for generating KC
MKK	The master key for generating KK

Kouichi Sakurai is Professor of Department of Computer Science and Communication Engineering, Kyushu University, Japan since 2002. He received B.E., M.E., and D.E. of Mathematics, Applied Mathematics, and Computer Science from Kyushu University in 1982, 1986, and 1993, respectively. He is interested in cryptography and information security. He is a member of IPSJ, IEEE and ACM.