Project Summary

The *digital provenance* of a digital object gives a history of its life cycle including its creation, update, and access. It thus provides meta-level information about the sequence of events that lead up to the current version of the object, as well as its chain of custody. Such provenance information can be used for a variety of purposes, such as identifying the origins of a document, assessing the quality or reliability of data, and detecting undesirable actions such as forgery or unauthorized alteration of data. However, all of these practical uses of provenance information presuppose that the provenance system is secure, reliable, efficient, and usable. For example, if it is possible to tamper with the provenance, it can cause users to get an incorrect impression of the authenticity or reliability of the underlying data, potentially with significant real-world consequences. Furthermore, these properties are not independent of each other: for example, collecting provenance at a fine granularity has the potential to make the system more useful, but can severely impact efficiency.

Intellectual Merit

A provenance-enabled system contains functions for *collecting*, *storing*, *validating*, and *querying* provenance. To be practical, these functions must be *secure*, *reliable*, *efficient*, and *usable*. To achieve these properties, the work proposed here builds on previous research on secure provenance, but does so by assuming a more liberal, and realistic, attack model. Previous research has assumed a *benign user* of provenance; this is a mistake since the creators of an artifact are often the ones most motivated to mislead others about its history. In the work proposed here we will protect against *insider attacks*, assuming all users are potentially malicious. Furthermore, we will address an important aspect of secure provenance, ignored by previous research, namely how to securely trace the *chain of custody* of a document that leaks outside the system. We will therefore investigate these closely interlinked topics:

- to mitigate insider attacks against provenance collection we will investigate novel techniques for *continuously updatable software tamperproofing*;
- to mitigate leakage of documents we will investigate novel techniques for *robust, continuous marking, collusion-free, text fingerprinting*;
- to allow for efficient storage and access of fine-grained provenance data we will investigate techniques for *anonymous storage on untrusted storage servers*;
- to assess the security of the system, we will conduct several red-team evaluation studies and "capture-the-flag" exercises.

Broader Impact

The broader impacts of the proposed research are fourfold. First, the exponential increase in production and sharing of digital content (documents, data, multimedia) has resulted in a demand for provenance tracking. Users often are not able to distinguish original from copied content, track ownership, ensure proper attribution or even simply know whether or not to trust the data that is being produced and disseminated. Our work has the potential to foster many different kinds of content sharing applications by providing a robust provenance management system design. It will advance our understanding of end-to-end security and access control in the context of provenance collection. Secondly, it will demonstrate the design of a secure, reliable, and usable provenance management system working on commonly employed office documents — such a system would put reliable tracking of document histories into the hands of scientists and researchers, government agencies, and corporations. Third, the educational efforts related to this research will lead to the development of broad and innovative courses on the topic of applied information security and digital intellectual property management. Finally, graduate and undergraduate students will be involved in all aspects of the research activities as an integral part of the project. We have a track record of involving undergraduates and members of underrepresented minorities in our research; this project will continue this tradition.

Key Words: provenance; software protection; tamperproofing; text fingerprinting.

TABLE OF CONTENTS

For font size and page formatting specifications, see GPG section II.B.2.

*Proposers may select any numbering mechanism for the proposal. The entire proposal however, must be paginated. Complete both columns only if the proposal is numbered consecutively.

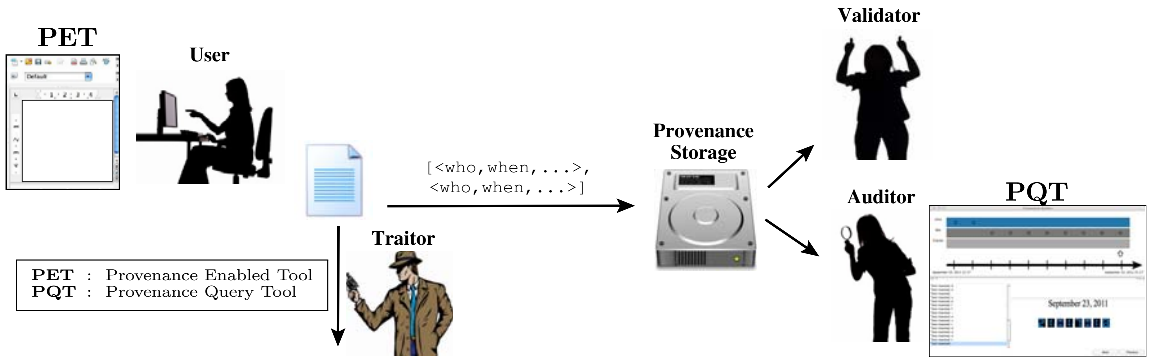

PET
User
Validator
Provenance Storage
[<who,when,...>, <who,when,...>]
Auditor
PQT
Traitor
PET : Provenance Enabled Tool
PQT : Provenance Query Tool
September 23, 2011

Figure 1: Principals and tools in a provenance-enabled system.

1 Introduction

There are a multitude of situations where one would like to know the history of a digital object: who created it, who modified it, when and where it was modified, in whose custody it has been since its inception, and so on. To illustrate, consider the following scenarios. First, consider a scientist who, upon reading a scientific paper, questions its conclusions. She could request the raw collected data from the authors, but without a complete history of how the data was collected and the exact sequence of transformations it has gone through, she will not be able to verify the results. Second, consider a law-enforcement agency that has taken photographs as part of a surveillance effort. Without being able to prove that the photographs were, in fact, taken at the place and time claimed and have not been doctored in any way, they may not be able to use them as a part of a criminal prosecution. Third, consider a group of authors collaborating on a manuscript. Without knowing who modified which part of the manuscript they will not be able to measure each person's contribution in order to apportion royalties. Fourth, consider an accountant performing a financial audit of a major corporation. Without being able to verify when and by whom book entries were modified, he will not be able to trace any irregularities in the accounts. And fifth, as a final scenario, consider a mobile phone manufacturer who finds the secret blueprints of their yet-to-be-released model published in a trade journal. Without being able to trace the schematics back to the insider (the "traitor") who divulged the trade secrets, they cannot stop any further leaks.

The *digital provenance* (DP) of a digital object gives a history of its creation, update, and access. It thus provides meta-level information of the sequence of events that lead up to the current version of the object, as well as its chain of custody. In particular, the meta-information describes the *what*, *where*, *when*, *how*, *who*, *which*, and *why* of the document's history [76]. "What" is the sequence of events that affect the data object. "When" and "where" record the time and location of a provenance event, respectively. "How" records the actions that lead up to an event. "Who" and "which" record the entities (individuals and organizations) effecting the event and the (software) tools used in the events, respectively. "Why", finally, describes the decision rationale behind the event.

Principals Conceptually, the principals of a provenance-enabled system are *users* who create and modify digital objects, *auditors* who query the provenance of an object, and *validators* who check the validity of an object's provenance. This is illustrated in Figure 1. In practice, the same person can serve in different roles at different times, and auditors and validators can be automatic services rather than individuals. Users employ *Provenance Enabled Tools* (**PET**s), to create and modify objects and auditors use *Provenance Query Tools* (**PQT**s) to access DP information in order to learn about an object's history. Also part of a provenance-enabled system is a storage sub-system for provenance and the digital objects themselves.

For concreteness, our work will focus on protecting provenance collected from office-like documents, such as texts, spreadsheets, drawings, and presentations. These types of documents are ubiquitous in the world of business, academia, and government, and being able to securely collect and manage their provenance would be hugely advantageous. The ideas proposed here carry over to other kinds of digital artifacts as well.

1.1 Security and Usability Goals

Provenance-enabled document-processing systems have multiple goals related to the security and privacy of the provenance and the documents themselves:

Confidentiality of Documents and Provenance. The system should protect the confidentiality of document contents from outside attackers, malicious insiders, unauthorized users, storage servers, and auditors. Sensitive provenance information also must not be disclosed. For example, we may want to keep the identities of users requesting copies of the document confidential from outsiders and storage servers.

Access Control. To be useful to users and auditors alike, tools for collection and querying of document histories should be an integral part of a complete document storage and retrieval system. In particular, access control and provenance are intimately linked: the system must only allow authorized users to modify documents, and only allow authorized auditors to access provenance data. Auditors may also be restricted to access specific documents and/or specific types of provenance data.

End-to-end Provenance Integrity. In general, a provenance system is useful only to the extent that it is trusted. The design of a provenance system must therefore assume that any principal can be malicious, and that any point on the chain from provenance collection to storage to auditing can be vulnerable. Thus, to produce reliable information, the system must ensure end-to-end integrity of provenance metadata.

While security and privacy are the main goals of the work proposed here, they cannot be examined in isolation. Provenance-enabled document-processing systems must also meet goals related to functionality, reliability, and efficiency:

Comprehensive Provenance Collection. The system must keep accurate provenance records for every event that affects a document. This includes *access* events (a user requesting a document), *editing* events (a user modifying a document), *disclosure* events (a traitor leaking a document), as well as *auditing* events (an auditor requesting a document's provenance, also called a *meta-provenance* event).

Usable Auditing. The system must keep comprehensive and fine-grained provenance metadata since it cannot be known *a priori* which historical information an auditor might need. Equally important, auditors should be able to query the provenance information in a natural and efficient manner without requiring programming or database expertise.

Reliability, Performance, and Scalability. The system should store documents and provenance data in a reliable way. With fine-grained provenance collection data can accumulate rapidly, and hence store and retrieve operations must have acceptable performance.

Interaction with Non-Provenance Enabled Tools. A provenance-enabled system has to be flexible, allowing users to employ familiar tools outside of the system in constructing their documents. Unfortunately, accurate metadata cannot be collected if a document escapes the provenance system, for example by being mailed by a traitor using a non-provenance enabled email client. To mitigate such leakage, it is desirable for a provenance enabled system to provide mechanisms to trace the source of the leaked information.

1.2 Insider Threat Model

In the research proposed here, we will investigate attacks against provenance systems, and defenses against such attacks. In our scenario, an adversary is motivated to perform operations on a document without this being reflected in the provenance metadata or, equivalently, modify provenance data to hide the real evolution of a document. For example, a malicious user might want to change the *who*-provenance of a document to implicate another user, or the *when*-provenance to hide when a modification was made, or make a modification to the document itself without this being accurately reflected in the *what*-provenance. If this kind of tampering is possible it can cause us to draw incorrect inferences about the authenticity or reliability of the underlying data, potentially with significant real-world consequences.

This attack scenario is an *insider attack* variant. Previous provenance-enabled systems have instead assumed a benign user; this is a mistake since the creators of an artifact are often the ones most motivated to mislead others about its history. In the work proposed here we will assume users are potentially malicious.

Provenance research often focuses on keeping accurate records of the *changes* that were made to a document, ignoring an equally important aspect of document's history, namely its chain of *custody.* This means that document *leakage* is a serious insider attack in a provenance-enabled system. For example, in Figure 1 the *traitor* leaks an object outside of the system without the appropriate provenance being collected. Such leakage occurs whenever the user prints a document, takes a screen shot, performs a copy-paste operation (into or out of the document), exports the document to a different format, or emails the document, if the provenance is not augmented with the corresponding change-of-custody information.

The document and provenance storage sub-system in Figure 1 is also susceptible to insider attacks from a malicious system administrator.

An important aspect of a provenance-collection system is the *granularity* at which provenance is gathered and stored. Previous systems have been coarse-grained, collecting information at the level of I/O-operations. Noticing that the difference between the antonyms "friend" and "fiend" is a single character, it is clear that in many legal, business, or military situations, provenance needs to be collected at the finest possible granularity, to prevent an adversary from making small, but significant, modifications without this being reflected in the provenance.

1.3 Objectives and Intellectual Merits

A provenance-enabled system contains functions for *collecting*, *storing*, *validating*, and *querying* provenance. To be practical, these functions must be *secure*, *reliable*, *efficient*, and *usable.* These goals are not independent of each other, and may involve tradeoffs. For example, while it is desirable to collect provenance at a fine level of granularity, the resulting large volume of provenance data can adversely impact efficiency. Similarly, security may be at odds with usability: in an insider-scenario such as this one, protecting against the actions of malicious users should not inconvenience benign users.

The research proposed here will investigate the design and implementation of a provenance management system that simultaneously addresses all these closely interlinked topics:

- to ensure end-to-end integrity of provenance data we will investigate novel techniques for *continuously updatable software tamperproofing* (Section 3.3);
- to mitigate leakage of documents we will investigate novel techniques for *robust, continuous marking, collusion-free, text fingerprinting* (Section 3.4);
- to allow for efficient storage and access of fine-grained and potentially large provenance data we will investigate techniques that will allow *anonymous storage on untrusted storage servers* (Section 3.5);
- to assess the security of the system, we will conduct several evaluation studies using a variety of users, including students in undergraduate and graduate security courses, scientists collaborating on research projects, as well as adversaries recruited from the Internet, enticed by "capture-the-flag" exercises (Section 3.7).

The outcome of the research proposed here will include three artifacts that will be made available to the research community to aid future secure provenance research: a) an application-level secure provenance library that can be easily integrated into new applications to experiment with new designs and algorithms; b) a fully functioning provenance-enabled system for office documents (based on the OpenOffice suite of tools), suitable for usability and practical security studies; and c) an analysis of data collected during exercises, allowing us to understand how attacks are carried out in practice.

1.4 Broader Impacts

The broader impacts of the proposed research are fourfold. First, the exponential increase in production and sharing of digital content (documents, data, multimedia) has resulted in a demand for provenance tracking. Users often are not able to distinguish original from copied content, track ownership, ensure proper attribution or even simply know whether or not to trust the data that is being produced and disseminated. Our work has the potential to foster many different kinds of content sharing applications by providing a robust provenance management system design. It will advance our understanding of end-to-end security and access control in the context of provenance collection. Secondly, it will demonstrate the design of a secure, reliable, and usable provenance management system working on commonly employed office documents — such a system would put reliable tracking of document histories into the hands of scientists and researchers, government agencies, and corporations. Third, the educational efforts related to this research will lead to the development of broad and innovative courses on the topic of applied information security and digital intellectual property management. Finally, graduate and undergraduate students will be involved in all aspects of the research activities as an integral part of the project. We have a track record of involving undergraduates and members of underrepresented minorities in our research; this project will continue this tradition.

2 Background and Related Work

2.1 Secure Provenance Collection

Sprov [42] is an application-layer C library that replaces `stdio.h` in a Unix application. The library captures all I/O write requests and appends them to the provenance chain. Documents and provenance chains are both stored on the user's machine and can be transferred together to other machines. One or more validators (called *auditors*) can verify the integrity of provenance chains. Sprov uses encryption, signatures, and checksums to ensure that adversaries cannot forge provenance chains (by inserting or deleting provenance records) that match an illicitly constructed document.

A malicious Sprov user can simply tamper with the library code itself in order to circumvent the provenance collection. Thus, Sprov is not secure against insider attacks. Operations such as printing or exporting of a document are not tracked. Thus, Sprov does not support traitor-tracing of documents that have escaped the system. Sprov's provenance collection is semantics-agnostic and of coarse granularity (raw I/O events). Thus, detailed information (such as time) about individual edit operations is not tracked.

Like Sprov, PASS, the *Provenance Aware Storage System* [61], collects I/O events as provenance, but does so at the Linux kernel level. Also like Sprov, PASS is susceptible to attacks by malicious users: no protection of the PASS kernel extension itself is provided, allowing adversaries to circumvent it by, for example, simply loading different kernel modules.

In the work proposed here, we will use the cryptographic techniques from the Sprov system to chain together provenance records, ensuring the integrity of the chain of events. We will extend this with *dynamic software protection* [22] techniques to protect the provenance-collecting code itself from insider attacks.

2.2 Traitor Tracing

An important and often overlooked aspect of digital provenance is the *chain of custody* of an artifact. By definition, once a digital object escapes our provenance system (through printing, export to a different format, etc.), we are no longer in control of it, and any further movements or edit operations will not be captured in our chain of provenance events. In this proposal we will explore the idea of *continuous document fingerprinting* to mark documents with the identity of the user editing it, allowing for leaked documents to be traced back to their owners.

A number of steganographic techniques have been proposed for watermarking of structured documents, such as texts. Common ideas include embedding identifying data in character-, word- or line-spacing [9], in the luminance of characters [8,83,84], in the color of characters [37], in synonyms, in visually imperceptible dots [51], in the shape of characters [82], and in grammatical structures [2,3]. These algorithms display a trade-off between robustness, embedding rate, and unobtrusiveness. For example, embedding in grammatical structures is robust to many attacks, but has a low embedding rate. Robustness also depends on the format in which a document is distributed: an adversary can more easily manipulate the line-spacing in a document distributed in PDF, PostScript, or XML [10] form than in one distributed as a bitmap, such as JPG. Robustness, embedding rate, and security can be improved by embedding multiple copies of a mark, by compressing and encrypting marks prior to embedding, and by standard error-correction techniques [13].

In this proposal we will devise a generic method that allows for multiple simultaneous watermark carriers in structured documents, improving on robustness. Our techniques will also improve on previous work by being resistant to *collusive attacks* [39]; i.e. computing the difference between two closely related, and differently fingerprinted, versions of the same document should not reveal the location of marks.

2.3 Scientific Workflow Provenance

In this proposal we are specifically concerned with the security and privacy of the provenance of office-type documents. Provenance collection and querying has seen applications in other domains as well. In *Scientific Workflow Provenance* [35], records are kept with detailed information about parameter settings and the sequence of processing steps (both experimental and computational) that have produced a particular data item. The provenance can later be queried to evaluate the validity and reliability of the data, to reproduce or debug the exact steps that produced the data, or to identify downstream results that might have been affected by an erroneous data item.

There are several tools for capturing scientific workflows, for example myGrid/Taverna [67], Kepler [1], and VisTrails [40]. The Open Provenance Model (OPM) [60] is a standard representation for workflow provenance.

Privacy and security is important also for scientific workflow provenance. Data from medical trials, for example, can contain sensitive information, and such information could be inferred from leaked fine-grained provenance. There are also intellectual property issues, particularly in the commercial arena, such as the pharmaceutical industry. While researchers may be willing to make some data available for public consumption, they may prefer to keep some private for proprietary reasons. This includes the data itself, but also the structure of the workflow, the modules that are a part of it, and any module parameter settings.

It is clear that the insider scenario investigated in this proposal applies to scientific provenance as well. A dishonest researcher who fabricates or falsifies data or results will want to cover his or her tracks by manipulating the provenance to conform to the doctored data. Similarly, if provenance is not properly safeguarded, proprietary information could be leaked by a malicious researcher to competitors or foreign governments.

2.4 Software Protection

A serious form of insider threat appears in any scenario where a malicious user is in complete control over a device and the software it contains, and can reverse engineer and tamper with it at will. Systems vulnerable to such attacks include smart meters in the *Advanced Metering Infrastructure (AMI)* [18,31,36,69], computer game clients [4,48,79,80,86], *Digital Rights Management* (DRM) systems, sensors in a Wireless Sensor Network, end hosts in *Electronic Medical Records* (EMR) systems, etc.

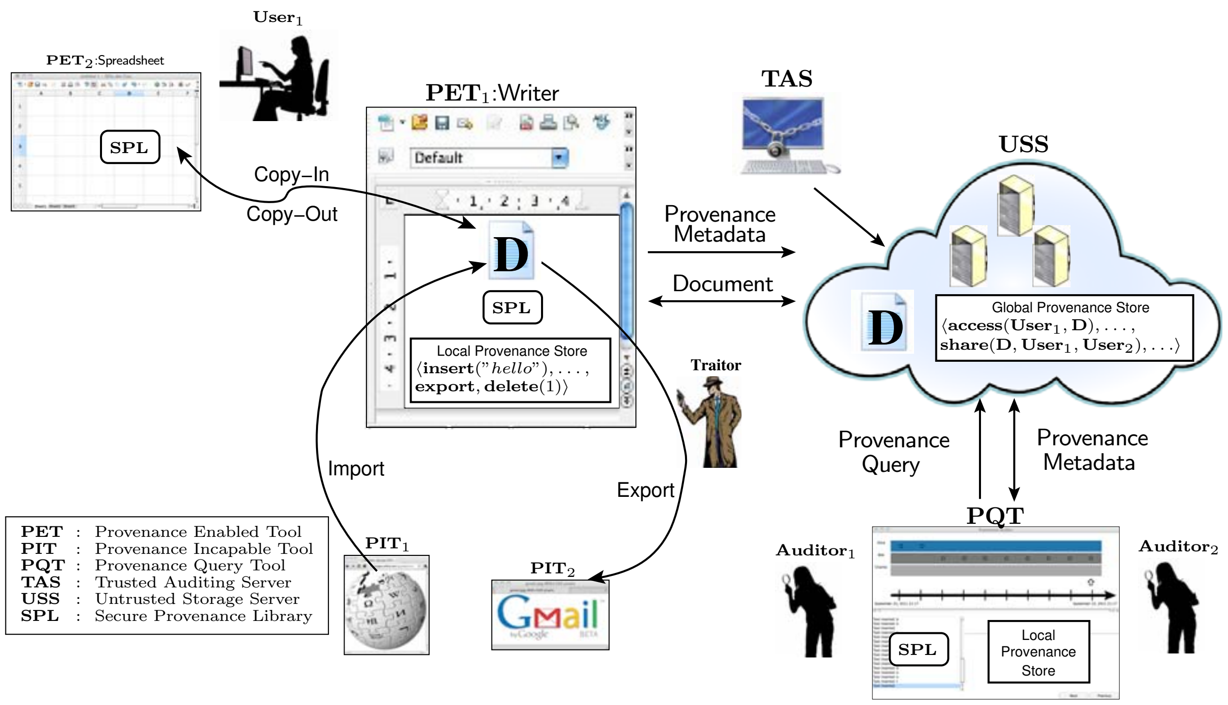
Insider attack-models must be very liberal to be realistic. It is expected that the adversary has many powerful reverse-engineering tools in his tool-chest, including *static analysis* (for disassembly, decompilation, slicing, etc.), *dynamic analysis* (for debugging, tracing, etc.), and *hardware attacks* to bypass hardware tamper-protection either through *non-invasive attacks* (such as side-channel attacks) or *invasive attacks* (such as decapsulating and probing the surface of a chip containing a secret cryptographic key).

The terms *software protection* [24] or *anti-tamper* refer to any technique for protecting against these types of attacks. Real-world solutions usually combine various forms of code obfuscation [5], tamperproofing [12,49], software watermarking [19], and birthmarking [65]. Since the device under attack is under complete control of the adversary it is expected that, given enough time and effort, all defenses will be breached. Continuously updatable software protection [22] has been suggested as a way to extend the lifetime of the protection.

2.5 Efficient Storage and Querying

Rozsnyai et al. [78]'s *store everything, discover later*-principle can lead to large amounts of metadata being collected, which causes them to consider cloud-based storage (using HBase/Hadoop, an open source MapReduce framework). They do not address the security implications of storing provenance in the cloud.

The SPADE [41] system collects provenance for distributed systems. Metadata is collected by monitoring system calls, paying special attention to data and computation that flow across network connections. In



User$_1$
PET$_2$:Spreadsheet
SPL
Copy–In
Copy–Out
PET$_1$:Writer
D
SPL
Local Provenance Store
⟨insert("hello"), . . . , export, delete(1)⟩
TAS
USS
Provenance Metadata
Document
D
Global Provenance Store
⟨access(User$_1$, D), . . . , share(D, User$_1$, User$_2$), . . .⟩
Traitor
Import
Export
Provenance Query
Provenance Metadata
PQT
PIT$_1$
PIT$_2$
Auditor$_1$
Auditor$_2$
SPL
Local Provenance Store
PET : Provenance Enabled Tool
PIT : Provenance Incapable Tool
PQT : Provenance Query Tool
TAS : Trusted Auditing Server
USS : Untrusted Storage Server
SPL : Secure Provenance Library

Figure 2: Overview of proposed secure digital provenance system.

SPADE, each host keeps the provenance metadata collected on it, stored in a relational database. Thus, while files are free to move from host to host, metadata is kept on the host on which it was collected. Not moving provenance data to a central location saves on network transfers (metadata can grow much larger than the document itself) and gives users control over the data they create (important, for example, in health-care applications). Queries, however, can now become more expensive, but judicious use of caching can relieve that.

3 Research Effort

We will next describe the architecture of our system (Section 3.1) and the overall objectives and research challenges addressed by this proposal (Section 3.2). We will then, in more detail, discuss the techniques for integrity, traitor-tracing, and anonymity (Sections 3.3, 3.4, and 3.5), we propose to investigate the results of a preliminary study (Section 3.6), and how the success of the research will be evaluated (Section 3.7).

3.1 Architectural Design

Figure 2 illustrates our proposed architecture for a secure provenance-enabled document management system. Users employ **PET**s to create and edit documents and auditors employ **PQT**s to query the resulting provenance. The *Trusted Auditing Server* (**TAS**) is operated by the users' organization, and is in charge of maintaining user identities and validating the provenance information. A **USS** is an untrusted storage server for provenance of documents. In practice, the **TAS** and the **USS** may be one and the same, but our design allows for expensive operations (such as storage of fine-grained, and hence large, provenance) to be outsourced to, say, honest-but-curious cloud storage providers. **PIT**s are tools outside the provenance system with which the user wants to exchange document data. The final component is the *Secure Provenance Library* (**SPL**) which is integrated into **PET**s and **PQT**s to collect provenance in a manner that guarantees authenticity and integrity.

Figure 3 shows how the OpenOffice Writer application has been turned into a **PET** by including our **SPL**. The **PET** maintains a copy of a document and a *local provenance store* (the provenance records of the document), until these are committed to the **USS**. The **SPL** monitors user actions and generates

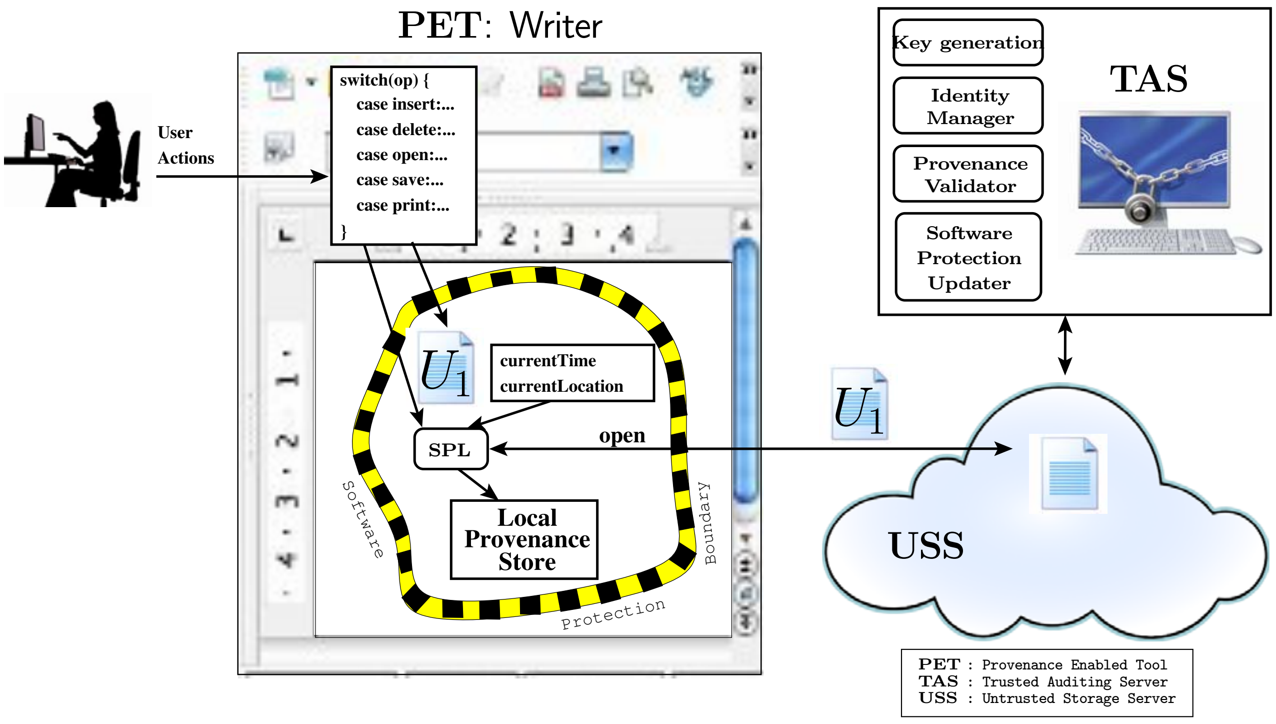


Figure 3: Overview of the **PET** and its interaction with the **TAS** and **USS**s.

the corresponding provenance records into the local provenance store. A typical provenance record will contain the user identifier, the current time, the current geo-location, the operation performed, and so on. Thus, in addition to the information it keeps internally, **SPL** has to acquire some environmental information (`currentTime`, `currentLocation`) from the operating system. On a save, **SPL** packages up the provenance records and the new document version, encrypts and signs them, and sends them to the **USS**.

3.2 Research Objectives and Challenges

In order to mitigate insider attacks in provenance-enabled systems we must address three fundamental challenges. The first challenge is to ensure the integrity of the entire provenance data processing chain. This means that the path from where an event is first collected (in our case, where the user enters data into an office document), to where it is processed (in our case, where provenance events are encrypted and chained together using checksums), to where it is stored (in our case, on untrusted storage servers) needs to protected from tampering. In practice this means that we must develop techniques for protecting the **SPL**'s code and data structures from modification. Without such protection, a malicious user or auditor would be able to subvert the **SPL** and perform editing or auditing actions without these being reflected in the provenance. Since no provably secure tamperproofing techniques exist, we will develop new techniques for continuously updatable software security to harden the **PET**s and **PQT**s. Previous work on updatable security has targeted code obfuscation [11,22,24] but that is not sufficient for our purposes; we need to develop updatable *tamperproofing* techniques.

The second challenge is to be able to trace documents that have escaped our system. For example, a malicious user may take a screenshot (using the built-in screenshot feature or, worse, an external camera) of the document being edited. In such cases, the proper provenance will not be collected, and we need to be able to trace the document back to the offending party. We will use text watermarking techniques to fingerprint office documents with the identity of users. However, current techniques are not sufficient for our purposes since a) they target static documents (our office documents are continuously changing as the user performs edit operations), b) they are not resistant to collusive attacks, and c) they are not robust against many simple transformations.

The third and final challenge is being able to use untrusted storage servers (such as cloud storage providers) to store the potentially fine-grained and, hence, large, provenance data. Unfortunately, current cryptographic techniques cannot be used since they leak user identities. We will investigate the use of novel signature schemes which allow using one verification key to validate signatures generated by different signers, ensuring anonymity.

3.3 Software Protection for Client-Side Integrity

In previous work (*Sprov* [42]), the integrity of a provenance chain for an artifact D is ensured by chaining together provenance records in (roughly) the following manner:

$$P_i = \langle \mathit{user}, W_i, \mathit{hash}(D_i), C_i \rangle \tag{1}$$

$$C_i = \mathit{signature}_{\mathit{user}}(\mathit{hash}(\mathit{user}, W_i, \mathit{hash}(D_i))|C_{i-1}) \tag{2}$$

Here, P_i is a provenance record for D, W_i is the modification that *user* has made, and $\mathit{hash}(D_i)$ is a hash of the current version of D. The $C_0, \ldots, C_n$ is a chain of checksums signed with the user's private key that ensures integrity.

In the Sprov system the checksums $C_0, \ldots, C_n$ are computed by a modified `stdio.h`, and as such easily compromised by a malicious user: simply replace `stdio.h` with a tampered version! As a result, the user could ensure that the provenance records indicate that a particular action was performed at a particular (wrong) time, at a particular (wrong) geo-location, or in a particular (wrong) order. Thus, not only must we keep the local provenance store encrypted and authenticated, it is also essential that we protect the integrity of the **SPL** code and data itself.

While many techniques have been proposed for tamperproofing software [12,24,49] they all suffer from the problem of offering only *time-limited protection* [6]. This is unacceptable in a system for secure provenance collection, where the veracity of the provenance could affect legal disputes, military and business decision-making, and verification of scientific results. In this proposal we will explore the idea of *continuously updatable tamperproofing* in order to extend the lifetime of the protection. This builds on and extends our previous work on updatable *code obfuscation* [22].

Research Challenges. The idea is for the trusted server, the **TAS**, to continuously generate new variants of the client's security-sensitive code (including the tamperproofing code), and force the clients (the **PET**'s and the **PQT**'s) to regularly accept these updates. The goal is to be able to generate and replace the code variants at a faster rate than the adversary can reverse engineer. There are several issues that need to be investigated:

1. **What tamperproofing algorithms are amenable to continuous updates?** Traditional introspection algorithms [12,49] add an intricate web of mutually dependent guards throughout the code, ensuring that removing any one guard will trigger another. This, however, also makes replacing guards difficult — in the worst case, updating a single guard could require the entire client program to be replaced, which would be unacceptable. There are also well-known attacks against such algorithms [6,85].
2. **What combination of granularity and frequency of updates achieve the right balance between security and performance?** The CodeBender [11] system replaces the entire program once per day, while the Tigress [22] system replaces individual functions multiple times per minute.
3. **How do we ensure that giving adversaries access to series of code variants does not leave the system open to collusive attacks?** If the only differences between two variants are the locations of the tamperproofing codes, then reverse engineering and removing that code will be trivial.

Proposed Research. We believe pure introspection tamperproofing techniques to be unsuitable for continuous replacement. Instead, we will investigate the use of *oblivious hashing* [14,50], techniques that check the *state* of the program for evidence of tampering, rather than checking the code itself.

One of the main problems in the software protection field is the lack of accurate attack models. Without such models it will be difficult for us to find the right balance between granularity and update frequency. To this end, we will instrument our system to collect data from actual attacks in the field, learning about the types of tools and techniques used by adversaries, and use this information to guide future development. Not only will this be useful to our own research, but it will help further software protection research as a whole.

To mitigate collusive attacks, we will investigate the use of various code obfuscation techniques to randomize the code between variants, in order to hide the actual differences. Such randomizations may have the undesirable side effects of making the program slower and larger, or making code updates slower. This work will build upon our previous work on the Tigress [22] system which contains a number of source-to-source obfuscating transformations.

3.4 Fingerprinting for Traitor-Tracing

In Figure 2 we show how a user can export a document **D** to a **PIT** tool (one that has not been provenance enabled) such as an email client. Similar export operations include printing, saving to a different format, copying part of the text into another document, etc. These types of operations must be supported by any practical document management system. In either case, the **PET** will add a record $\langle \texttt{User}_1, \mathbf{export}, \mathbf{D} \rangle$ to the provenance store. While nothing further can be said about subsequent whereabouts of **D**, at the very least we know from the provenance that at a particular time a particular document version was exported by a particular user.

There are several ways in which is a malicious insider could circumvent the collection of export provenance events. First, he could bypass the tamperproofing that protects the **SPL** (even given the techniques for continuous update of software protection code in Section 3.3, such protection can never be absolute). Second, he could make a series of screen captures of the **PET** using the operating system's built-in screenshot facility. Third, he could take a series of snapshots of the screen using an external camera. The ultimate export attack is to manually transcribe the document; for this attack there is no defense, and we will not consider it further.

Research Challenges. We will use *traitor-tracing* techniques to mitigate these situations. These mechanism will allow tracing the individual malicious user when an exposed document is detected.

The idea is to *mark* the document with an identifier unique to each user. Such marks are known as *fingerprints* or *watermarks*, and there exists many marking algorithms for different kinds of digital objects, such as images [33], video, audio, text [2], and software [24]. Text watermarking algorithms are described in Section 2.2. The fingerprint allows an exposed document to be traced back to the client who leaked it. Well-designed fingerprints should be resilient to document modifications, and even to (limited) collusion between clients.

While there has been much prior work on traitor tracing [17] through fingerprinting, our scenario is very different from what has been considered in the past, resulting in multiple challenges that require novel solutions.

A first challenge is that fingerprinting has to be performed by the **PET**, which is under the insider's control. This is different from standard traitor tracing scenarios where a trusted party marks the object before it is distributed to an untrusted party. Thus, not only does the integrity of the **PET** code need to be protected (as in Section 3.3) but also its *confidentiality.* An adversary who can "peek" into the **PET**, either statically or dynamically as a document is being fingerprinted could learn enough about the location of fingerprints to be able to remove them from the marked document.

A second challenge are collusive attacks. While studied extensively for image, audio, and video fingerprinting, we have been able to find no work on collusive-resistant text fingerprinting. There are several related attacks. First, the user could save the document twice, making minor edits between saves, and compare the resulting versions to determine the location of marks. Second, two colluding users could save the same document, compare them, and remove the marks based on the differences in their user identifiers. Finally, a malicious user who can get hold of a document saved by a second user can perform a *spoofing attack* by comparing the document to their own saved version, remove their own marks, and insert the marks of the second user.

A final challenge is the requirement that the system should be resilient against screenshot attacks, either by the built-in facility or by an external camera. Since the **PET** will only show a part of the document at any one time, the identity of a document must be extractable from any (small) part of the document. This is different from standard text fingerprinting scenarios where the entire document is available for extraction.

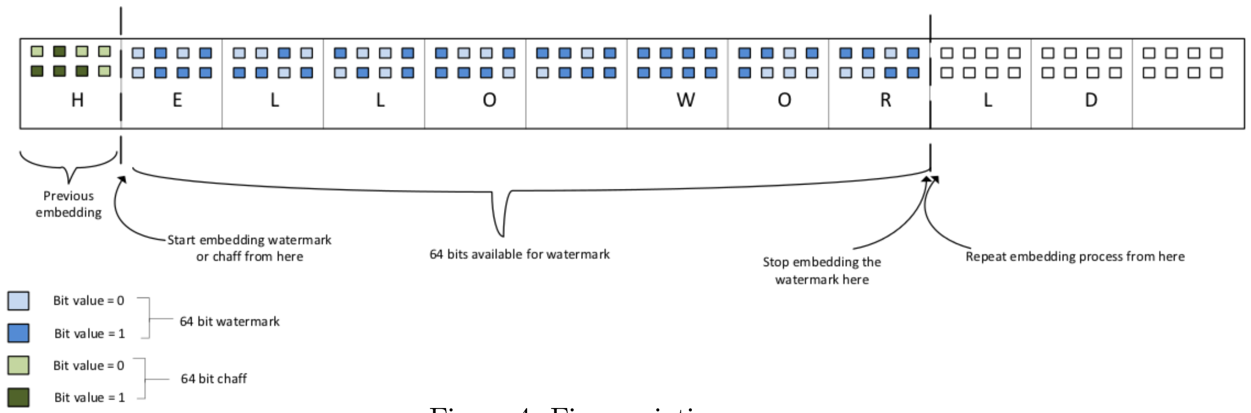

H
E
L
L
O
W
O
R
L
D
Previous embedding
Start embedding watermark or chaff from here
64 bits available for watermark
Stop embedding the watermark here
Repeat embedding process from here
Bit value = 0
Bit value = 1
64 bit watermark
Bit value = 0
Bit value = 1
64 bit chaff

Figure 4: Fingerprinting process.

Proposed Research. We propose to investigate a *generic, continuous marking, collusion-free* text marking algorithm. *Generic*, means that to strengthen marks against a variety of attacks, rather than focusing on a single embedding carrier (such as word-spacing), we allow any combination of carriers. *Continuous marking* means that, as the user is typing and editing the document, the view that is presented on the screen is continuously updated, ensuring that screenshots will always be marked. *Collusion-free* means that every single marking bit, from an attacker's point of view, is random, and gives away no information.

We sketch the idea next. From a PRNG seeded with the user's ID and the current document version number a sequence of random 56-bit watermark numbers is generated. To each number is appended an 8-bit error-correcting code, yielding a 64-bit number. We next slide a 64-bit window over the embeddable bit-positions in the current viewport. These embeddable bits could come from any combination of watermark carriers such as luminance, color, spacing, character shapes, synonyms, etc. (see Section 2.2). At each 64th bit position we flip a biased coin and either embed the next watermark number generated by the PRNG or 64 bits of random chaff. This process is illustrated in Figure 4.

As a result, a) multiple fingerprints are embedded in every section of the document allowing us to trace leakage of parts of documents; b) every carrier bit is random and collusive attacks are prevented; and c) multiple simultaneous de-watermarking transformations are necessary to perturb all watermark carriers, and robustness is improved.

If the number of users and documents is fairly small, an exhaustive scan over the document's embedded carrier bits and all generated watermark numbers will be sufficient to extract the fingerprint and identify the user who leaked a document. We will investigate faster algorithms (perhaps based on the Blum Blum Shub [7] PRNG which allows direct calculation of the *i*:th random number) for cases when this exhaustive method is too slow.

3.5 Anonymity

In order to successfully use untrusted clouds for document storage, we must ensure the anonymity of users and the integrity and confidentiality of documents and provenance. To accomplish this, we propose to combine two recent results, presented in Hasan, Sion, Winslett [42] (for integrity and confidentiality) and Lu, Lin, Liang, Shen [58] (for anonymity).

Research Challenges. Unfortunately we cannot directly use the Sprov scheme in equation 2 since a) it was originally designed for a scenario where provenance and documents follow each other and b) the signature $signature_{user}$ leaks the user's identity. Lu et al.'s [58] proposal, however, employs a special type of signature scheme which allows using one verification key to validate signatures generated by different signers. This property allows them to provide *anonymity* of the signers, using a trusted manager to maintain the keys and the corresponding identities of the clients. Unfortunately, this complicates the design and restricts it to specific cryptographic primitives.

Proposed Research. We propose to adapt *Sprov*'s cryptographic mechanisms to a scenario with honest-but-curious storage providers, such as current cloud storage servers. This will ensure integrity and confidentiality of provenance chains. We will furthermore combine this scheme with the signature scheme proposed by Lu et al. to ensure anonymity of users and auditors. A major challenge will be designing *efficient* protocols involving the **TAS**, **PET**s, **PQT**s, and **USS**s, all of whom have different levels of trustworthiness and performance characteristics.

3.6 Preliminary Study

The primary goal of the work proposed here is to design algorithms and systems that allow for end-to-end secure collection and processing of fine-grained provenance for documents. A secondary, but equally important goal, is that this processing is efficient enough to be unobtrusive; i.e. we want usable security.

In contrast to previous work (such as *Sprov* [42] and PASS [61] which collect raw I/O events as provenance), **SPL** will collect provenance for every user keystroke. This will entail a considerable amount of overhead. First, it is necessary to create and populate a provenance record for each keystroke. Each such provenance record then has to be encrypted before being written to the local provenance store. Furthermore, the code for the **PET** will incur some overhead from the software protection measures discussed in Section 3.3. On the other hand, we envision the primary use of the system to be interactive, which will limit the rate at which provenance events, i.e., keystrokes, occur. There are therefore two performance-related questions that will have to be addressed: first, are the overheads high enough to affect the usability of the system in an interactive setting? and second, to what extent can we mitigate these overheads, and what effect does this have on the collected provenance?

To verify the feasibility of our design, both from an engineering and a performance perspective, we implemented a simple prototype consisting of an OpenOffice plugin that listens to user events and stores each event in a list, then on each save of the document encrypts the collected provenance data and sends it to a server on Amazon's EC2 cloud. Ideally, each provenance event should correspond to a user keystroke (for entering text) or operations such as save, import, etc.; in our preliminary implementation, we used a script to echo the data into the provenance system, and the data rate from the script caused each provenance event to correspond to several characters. We tested our prototype with two inputs: the Declaration of Independence and Shakespeare's Hamlet. Our results are given below. Here, **Size** is the input document's size; N_{prov} is the number of provenance events recorded; **Time** and **Space** give the overheads for provenance collection. The last three columns of the table give per-event statistics for provenance data.

Input	**Size** (Sz) (bytes)	N_{prov}	**Time** (T) (sec)	**Space** (bytes)	Sz/N_{prov} (bytes)	T/N_{prov} (msec)	S/N_{prov} (bytes)
Decl. Indep.	24,265	568	39	765,012	42.7	68.7	1346.8
Hamlet	193,082	19,672	1,339	25,858,882	9.8	6.8	1314.5

It can be seen that each provenance event corresponds to between 10 and 43 characters read from the input and incurs a runtime overhead of between 7 and 68 msec, and that each provenance record takes approximately 1.3Kbytes of space. Interestingly, the per-event overheads decrease as the input file size and the number of provenance events increases, suggesting some amortization of overheads.

In our complete system there will be additional overheads not accounted for in the prototype: software tamperproofing code will be continuously updated, one provenance record will be generated per character typed (rather than one record per 10-43 characters, as in this experiment), and text will be continuously fingerprinted as the document is edited. Even so, we believe this simple experiment shows that, given careful design and optimization, interactive performance will be acceptable.

Not all scenarios will have the same needs for performance and security. Therefore, an important part of the work proposed here we will be to investigate the tradeoffs between granularity of provenance information, level of software protection, and the resulting runtime overhead.

3.7 Evaluation

The over-arching theme of this proposal is that in order for provenance data to be truly useful, consumers of provenance must have confidence in the complete chain that processed the data. This includes the initial collection of provenance events, encryption and local storage of provenance records, transfer to storage servers, processing by provenance query tools, etc. In other words, provenance is only as useful as our trust in its validity, and any weak link in the processing chain will weaken that trust. Previous work on secure provenance has failed to consider the integrity of the complete chain.

Furthermore, security cannot be considered separately from usability. There are three desirable properties related to the usability of provenance: the performance of provenance collection has to be good enough to be non-intrusive; the granularity of collected events has to be fine-grained enough that important information about the evolution of the document is not lost; and the query language has to be powerful enough that useful queries can be formed.

There are clearly trade-offs between usability and security: higher levels of software protection and more frequent and complex text fingerprinting will result in higher computational overhead, and this, in turn, will affect the level of provenance granularity that can be effectively collected. It is the goal of the evaluation part of this project to explore these trade-offs.

3.7.1 Evaluation of Client-Side Security

Since the user has complete control over the device on which the **PET** (or, for auditors, the **PQT**) is running, they are able to mount an insider attack on it. Theoretical evaluation of the strength of protection against such insider attacks is unlikely to prove very useful — we are aware of no detailed attack models against which the results of such evaluations could be validated. Instead, we are going to have to resort to very practical evaluation procedures.

As a first step, we will evaluate the strength of the client-side security by using student programming competitions in computer security classes to see how quickly, and in what ways, the continuously updatable tamperproofing can be broken. While experiments using students may not produce results that can be readily extrapolated to highly-skilled and experienced attackers, it will nevertheless help us identify and fix problems, and also suggest which aspects of our defenses are strong and which are easily broken.

As a second step, we will run capture-the-flag competitions [16] with attackers recruited from cracker communities. These adversaries are expected to be more sophisticated, and with more varied background, than the students from the previous exercise. The adversaries will download our version of OpenOffice that has been extended for secure provenance collection, and be asked to make a change to an existing document (such as changing "friend" to "fiend") or its provenance chain (such as changing the user who performed a particular edit, or the time a particular edit was made), without this being detected. Rewards will be given out for "fastest successful attack," "most clever hack," etc. The competition will be announced on popular cracker sites such as `http://crackmes.de`.

Our OpenOffice variant will be instrumented to collect information on how attacks were accomplished, including what tools (such as debuggers, virtual machines, tracers, decompilers, etc.) were used, the order in which attacks were attempted, the time spent on each attack, etc. The analyses of the collected data will not only be useful for evaluating and improving on our own software protection algorithms, but will be published to allow us and other researchers to build realistic attack models for future research into these types of attacks.

3.7.2 Evaluation of Traitor-Tracing

We will also evaluate the efficacy of our approach to traitor-detection via fingerprinting by experimenting with ways to defeat the fingerprinting scheme. The goal will be to export a provenance-enabled document to a non-provenance-enabled setting and then try to identify and/or remove the fingerprint.

We need to evaluate three aspects of the text fingerprinting algorithms we invent: their *robustness* against removal attacks, their *unobtrusiveness* to human observers, and their resilience against *collusive* attacks. A fourth aspect of fingerprinting, *embedding-rate* (the amount of information that can be embedded in a document of a particular size), is less important in our scenario than in traditional traitor-tracing since the number of users who would edit or export a document is typically small (on the order of dozens or at most hundreds).

With respect to robustness, we need to consider different types of document formats, including structured formats (such as OpenOffice' XML file format), page description formats (such as PDF and PostScript), and bitmap formats (such as JPG). Depending on the nature of the export operation, different file formats will be produced. The goal of our algorithm design is to combine multiple different embedding vectors (word-spacing, line-spacing, luminance, color, character shape, etc.) such that, even though each of these may be easily defeated for a particular format, their combination would be robust for all formats. This needs to be verified.

With respect to unobtrusiveness, we need to evaluate how combining multiple different embedding carriers affects human perception. While it is entirely possible that minor perturbations of line-spacing *or* word-spacing are imperceptible to the human visual system, this may not be the case when these embedding methods are combined.

With respect to collusive attacks, we need to evaluate whether our designs leak any information about user identities. In particular, the algorithm sketch in Section 3.4 is designed such that each embeddable bit (whether a part of a fingerprint or chaff) is completely random. This needs to be verified.

3.7.3 Evaluation of Usability

The **PET** overhead is a combination of the cost of collecting provenance events, encrypting and chaining together provenance records, protecting the **PET** code and data from insider attacks using continuously updated software protection, and continuously fingerprinting text to protect against document leakage. In practice, to evaluate client side (**PET**) performance, we need to determine the parameters (provenance granularity, tamperproofing code update frequency, fingerprinting frequency) for which we can *actually* collect provenance without interactive performance suffering so much that the system becomes unusable.

We also need to evaluate the performance of the traitor-tracing scheme, in particular the cost of extracting fingerprints from a leaked document encountered in the wild, and mapping these back to the offending user. While this is an operation that will, presumably, be performed infrequently, it still has to be reasonably efficient.

A secondary issue is provenance query (**PQT**) performance, which could be a concern since our system will generate large amounts of fine-granularity provenance data. We consider this question outside the scope of the proposed study.

4 Impact and Merit

Science impact: Provenance-enabled systems that are simultaneously secure, reliable, efficient, and usable have not been well studied. Our research will examine and prototype the algorithms and designs necessary for constructing such systems, and study the tradeoffs between security and usability. Our secure provenance prototype will be made publically available, providing a platform for other researchers to further study the security of provenance collection and processing. Our system will be built by extending the free and open-source OpenOffice suite of document management tools. Many organizations (including scientific, governmental, and commercial) require accountability from their constituents, and will benefit from being able to use our system to query the history of documents with a high degree of trustworthiness.

Educational Impact: All of the PIs are very concerned with education. We are actively involved in undergraduate research and have supported numerous undergraduate research assistants using NSF REU supplements as well as regular grant awards. We will continue this tradition through this project, e.g., by encapsulating specific tasks in this research effort into REU projects that will be offered to our undergraduates. Furthermore, we will involve our graduate students as mentors of these undergraduates, and in the construction of self-contained undergraduate research projects.

As part of this research, we will develop educational modules that can be incorporated into the computer security courses being developed as well as existing courses. An important component of these modules will be laboratory exercises that teach students the "nuts and bolts" of securing a computer and how to produce software that avoid introducing insider vulnerabilities. The PIs also plan to coordinate the development of a graduate seminar on software protection.

Dissemination: The results of this work will be disseminated in several ways, including publications in conferences, workshops, and journals, and by releasing tools to the community, as we have done in the past. We will also disseminate our work by preparing, on a yearly basis, research and education portfolios for distribution. The research portfolio will include items such as our published papers and technical reports, software tools developed during the project, tutorials, case studies, evaluation suites, lecture slides and class notes. The research and education portfolios will be advertised and made available at education conferences (SIGCSE and ITiCSE), and research conferences (CCS, DSN, USENIX, Oakland, etc.), and a yearly workshop, organized by PI Collberg. The first workshop was held in Beijing, China in July 2011 (see `http://www.ieeeisi.org/ssp_prog.html`). The 2012 workshop was co-located with ACM PLDI (see `http://profs.scienze.univr.it/ssp2012/SSP2012/Welcome.html`).

Technology Transfer: The PIs have substantial industry contacts and will continue to explore avenues for transitioning the results of this research to relevant companies so the benefit to the public is quickly realized. The PIs understand the role and importance of technology transfer. Consequently, the PIs are committed to this research having significant impact both in academia and industry. We will furthermore work with *Eller McGuire Entrepreneurship Program* to explore technology transfer opportunities for our provenance tracking and management system.

Diversity: While this research project will make contributions in educating students to appreciate the challenges of securing software, it will also contribute to diversity. All the PIs involved in this project have a strong commitment to include underrepresented students in the proposed work.

5 Preliminary Work

Collberg has designed high-level software protection algorithms, including algorithms for code obfuscation [27, 28,30], software watermarking [20,21,26,29,63,66], and birthmarking [64,65]. He is the author of the first comprehensive book on algorithms for software protection [24]. Debray's prior work related to this project includes the techniques for binary-level anti-disassembly obfuscation [54,55,70], low-level analysis of obfuscated code [32,52,81], and software protection and intrusion detection [56,59,72]. Ram's prior work on provenance management relevant to this project includes definition of the W7 model for capturing provenance [74–76], mechanisms for querying provenance [57] and tracking and harvesting provenance in scientific application [77].

6 Results from Prior NSF Support

Collberg was PI on NSF grant CCR-0073483, entitled *Code Obfuscation, Software Watermarking, and Tamper-Proofing — Tools for Software Protection*, $265,000, September 1, 2000–August 31, 2004. This project resulted in papers describing techniques for constructing error-correcting graphs [20], an overview of software protection techniques [30], a description of the SandMark software protection infrastructure [23], novel software watermarking algorithms [19], SandMark's obfuscation executive [47], and evaluation of published software protection techniques [21,25,63,66]. Three bachelor's, one master's, and one PhD [62] thesis were completed. Debray was PI on NSF grant CNS-0410918, *A Holistic Approach to Compiler-Assisted Optimization of Software Systems*, $568,000, Sept. 1, 2004–Aug. 31, 2009. Papers include binary-level code obfuscation and deobfuscation [70,81], low-level reverse engineering and malware analysis [32,52], software protection and intrusion detection [56,59,72], semantics-based malware detection [34], understanding dynamically modified code [38], and optimization of operating system kernels [44–46,68,73]. Four bachelor's (two by women), one master's [15], and two PhD [43,71] theses were completed. Ram was PI of NSF IIS-#0455993 *Investigating Data Provenance in the context of new product design and development,* $244,000 May 1, 2005-2009. This grant resulted in research papers on framework for defining provenance semantics [57,74–76], and a prototype system for provenance tracking and querying [77] and defining a structure for provenance for the iPlant collaborative project [77], two dissertations, and three MS theses. She has a current NSF grant (#DBI-0735191), "The iPlant Collaborative" and a part of her work in this project is to define structures for provenance tracking for plant biology scientific datasets. Collberg and

Debray were recently awarded NSF grant CNS-1145913 *EAGER: Man-at-the-End Attacks—Defenses and Evaluation Techniques*, $269,649, September 1, 2011–August 31, 2013.

7 Collaboration Plan

Since the proposed research cuts across provenance, software protection, and security analysis, expertise in each of these areas is needed. The assembled team is well qualified to carry out the proposed research with recognized strength in each of these areas: Debray is an expert in reverse engineering, security analysis, and performance analysis; Collberg is an expert in software protection, code obfuscation, and software fingerprinting; and Ram is an expert in semantics of provenance.

The investigators recognize that the success of this project depends on a fully integrated approach and are committed to working closely together. However, each PI will be responsible for and lead one aspect of the project: Debray will lead research into attack models and algorithms for high-performance provenance collection and storage; Collberg will lead research into security of provenance, including algorithms for traitor-tracing; Ram will lead research into usability of provenance.

Coordination and Integration. At the beginning of each year a two day mini-workshop will take place where students and faculty will give presentations and to which industry representatives will be invited. During these meetings the PIs will also identify specific research goals for the following year, the tasks that must be accomplished by each researcher, and the time frame. At the beginning of each term meetings will be held to discuss the goals and accomplishments of the previous term and set research targets for the next term. PIs and students will meet in monthly higher-level meetings that will focus on the overall progress of the project, track progress towards the goals set, identify technical issues, discuss approaches and solutions for them, and set particular research targets. More specific issues will be discussed in weekly meetings, typically held between a student and a PI with involvement by other students and/or PIs only as necessary. We will furthermore meet on an *ad hoc* basis to work on multi-author collaborative papers.

Development Plan and Milestones. The development plan for this project is as follows:

Year 1: Architecture: Extend the core of one OpenOffice tool to construct a **PET** that can collect provenance data. Design and prototype a **TAS/USS** that can maintain user identities and store provenance data. Security: Implement a code transformation framework (based on LLVM [53]) for building tamperproofing systems.

Year 2: Security: Design and implement oblivious tamperproofing algorithms, protecting the integrity of the **PET**, using the framework developed in Year 1, but without continuous updates. Develop algorithms to protect the anonymity of users, and integrate them into the **TAS**. Evaluation: Develop and run attack exercises in computer security classes.

Year 3: Security: Develop algorithms to extend the tamperproofing system from Year 2 to do continuous updates. Develop and prototype algorithms for document fingerprinting for traitor-tracing. Performance: Evaluate the performance of **PET**s, develop new algorithms as necessary to improve performance, and determine the finest grained provenance collection and highest level of software protection feasible. Evaluation: Deploy the **PET** on the web to collect information on attack patterns, improve the software protection and fingerprinting in response to knowledge gained.

Examples of expected Ph.D. projects include: (a) *Continuously updatable tamperproofing for secure provenance collection*, (b) *Traitor-tracing in a secure provenance system*, (c) *Software Protection Attack Models.*

References

[1] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. In L. Moreau and I. T. Foster, editors, *IPAW*, volume 4145 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2006. 5

[2] M. Atallah, V. Raskin, C. Hempelmann, M. Karahan, R. Sion, and K. Triezenberg. Natural language watermarking and tamperproofing. In *Proc. 5th International Information Hiding Workshop*, 2002. 4, 9

[3] M. J. Atallah, V. Raskin, M. Crogan, C. Hempelmann, F. Kerschbaum, D. Mohamed, and S. Naik. Natural language watermarking: Design, analysis, and a proof-of-concept implementation. In *IHW '01: Proceedings of the 4th International Workshop on Information Hiding*, pages 185–199, London, UK, 2001. Springer-Verlag. 4

[4] S. Balfe and A. Mohammed. Final fantasy - securing on-line gaming with trusted computing. In B. Xiao, L. T. Yang, J. Ma, C. Müller-Schloer, and Y. Hua, editors, *ATC*, volume 4610 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2007. 5

[5] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs (extended abstract). In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, 2001. LNCS 2139. 5

[6] P. Biondi and F. Desclaux. Silver needle in the skype. In *Black Hat Europe*, Feb-Mar 2006. `www.blackhat.com/presentations/bh-europe-06/bh-eu-06-biondi/bh-eu-06-biondi-up.pdf`. 8

[7] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 15(2):364–383, 1986. 10

[8] P. V. K. Borges and J. Mayer. Text luminance modulation for hardcopy watermarking. *Signal Process.*, 87(7):1754–1771, July 2007. 4

[9] J. Brassil, S. Low, and N. Maxemchuk. Copyright protection for the electronic distribution of text documents. *Proceedings of the IEEE*, 87(7):1181–1196, 1999. 4

[10] A. Castiglione, B. DAlessio, A. Santis, and F. Palmieri. Hiding information into ooxml documents: New steganographic perspectives. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 2(4):59–83, 2011. 4

[11] M. Ceccato and P. Tonella. Codebender: Remote software protection using orthogonal replacement. *IEEE Software*, 28(2):28–34, 2011. 7, 8

[12] H. Chang and M. J. Atallah. Protecting software code by guards. In *Security and Privacy in Digital Rights Management, ACM CCS-8 Workshop DRM 2001*, Philadelphia, PA, USA, Nov. 2001. Springer Verlag, LNCS 2320. 5, 8

[13] Q. Chen, Y. Zhang, L. Zhou, X. Ding, and Z. Fu. Word text watermarking for ip protection and tamper localization. In *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on*, pages 3595–3598. IEEE, 2011. 4

[14] Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. Jakubowski. Oblivious hashing: A stealthy software integrity verification primitive. In *5th Information Hiding Workshop (IHW)*, pages 400–414, Oct. 2002. Springer LNCS 2578. 8

[15] M. M. Chhabi. Efficient taint analysis using multicore machines. M.S. Thesis, Department of Computer Science, The University of Arizona, Tucson, May 2007. 14

[16] N. Childers, B. Boe, L. Cavallaro, L. Cavedon, M. Cova, M. Egele, and G. Vigna. Organizing Large Scale Hacking Competitions. In *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, Bonn, Germany, July 2010. 12

[17] B. Chor, A. Fiat, and M. Naor. Tracing traitors. In *Proceedings of the 14th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '94, pages 257–270. Springer-Verlag, 1994. 9

[18] T. Claburn. Smart grid lacks smart security. *InformationWeek*, March 24 2009. 5

[19] C. Collberg, E. Carter, S. Debray, J. Kececioglu, A. Huntwork, C. Linn, and M. Stepp. Dynamic path-based software watermarking. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 04)*, 2004. 5, 14

[20] C. Collberg, E. Carter, S. Kobourov, and C. Thomborson. Error-correcting graphs for software watermarking. In *Workshop on Graphs in Computer Science (WG'2003)*, June 2003. 14

[21] C. Collberg, A. Huntwork, E. Carter, and G. Townsend. Graph theoretic software watermarks: Implementation, analysis, and attacks. In *Workshop on Information Hiding*, pages 192–207, 2004. 14

[22] C. Collberg, S. Martin, J. Myers, and J. Nagra. Distributed application tamper detection via continuous software updates. In *ACSAC*, 2012. 4, 5, 7, 8, 9

[23] C. Collberg, G. Myles, and A. Huntwork. Sandmark–a tool for software protection research. *IEEE Security and Privacy*, 1(4):40–49, 2003. 14

[24] C. Collberg and J. Nagra. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection.* Addison-Wesley Software Security Series. Addison-Wesley, July 2009. 5, 7, 8, 9, 14

[25] C. Collberg and T. R. Sahoo. Software watermarking in the frequency domain: implementation, analysis, and attacks. *J. Comput. Secur.*, 13(5):721–755, 2005. 14

[26] C. Collberg and C. Thomborson. Software watermarking: Models and dynamic embeddings. In *In Conference Record of POPL '99: The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (Jan. 1999)*, 1999. 14

[27] C. Collberg, C. Thomborson, and D. Low. Breaking abstractions and unstructuring data structures. In *IEEE International Conference on Computer Languages 1998, ICCL'98.*, Chicago, IL, May 1998. 14

[28] C. Collberg, C. Thomborson, and D. Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages 1998, POPL'98*, San Diego, CA, Jan. 1998. 14

[29] C. Collberg, C. Thomborson, and G. Townsend. Dynamic graph-based software fingerprinting. *TOPLAS*, 29(6):1–67, Oct. 2007. 14

[30] C. S. Collberg and C. Tomborson. Watermarking, tamper-proofing, and obfuscation – tools for software protection. *IEEE Transactions on Software Engineering*, 8(8), 2002. 14

[31] M. Conner. Tamper-resistant smart power meters rely on isolated sensors. *Design News*, Aug. 2009. `http://www.designnews.com`. 5

[32] K. Coogan, S. Debray, T. Kaochar, and G. Townsend. Automatic static unpacking of malware binaries. In *Proc. 16th. IEEE Working Conference on Reverse Engineering*, Oct. 2009. To appear. 14

[33] I. J. Cox, M. L. Miller, and J. A. Bloom. *Digital Watermarking: Principles and Practice*. Morgan Kaufmann, 2002. 9

[34] M. Dalla Preda, M. Christodorescu, S. Jha, and S. Debray. A semantics-based approach to malware detection. *ACM Transactions on Programming Languages and Systems*, 30(5):1–54, Aug. 2008. 14

[35] S. Davidson, Z. Bao, and S. Roy. Hiding data and structure in workflow provenance. In *Proceedings of the 7th international conference on Databases in Networked Information Systems*, DNIS'11, pages 41–48, Berlin, Heidelberg, 2011. Springer-Verlag. 5

[36] M. Davis. Smartgrid device security — adventures in a new medium, 2009. `http://www.blackhat.com/presentations/bh-usa-09/MDAVIS/BHUSA09-Davis-AMI-SLIDES.pdf`. 5

[37] M. Du and Q. Zhao. Text watermarking algorithm based on human visual redundancy. *AISS: Advances in Information Sciences and Service Sciences*, 3(5):229–235, 2011. 4

[38] B. Dux, A. Iyer, S. K. Debray, D. Forrester, and S. Kobourov. Visualizing the behavior of dynamically modifiable code. In *Proc. 13th. IEEE International Workshop on Program Comprehension*, May 2005. 14

[39] F. Ergun, J. Kilian, and R. Kumar. A note on the limits of collusion-resistant watermarks. In *Advances in Cryptology, Eurocrypt '99*, volume 1592, pages 140–149, 1999. 5

[40] J. Freire, C. T. Silva, a. E. S. Steven P. Callahan, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *Proceedings of the 2006 international conference on Provenance and Annotation of Data*, IPAW'06, pages 10–18, Berlin, Heidelberg, 2006. Springer-Verlag. 5

[41] A. Gehani, M. Kim, and T. Malik. Efficient querying of distributed provenance stores. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 613–621, 2010. 5

[42] R. Hasan, R. Sion, and M. Winslett. The case of the fake picasso: Preventing history forgery with secure provenance. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, 2009. 4, 8, 10, 11

[43] H. He. *Memory Footprint Reduction of Operating System Kernels*. PhD thesis, Dept. of Computer Science, University of Arizona, Mar. 2009. 14

[44] H. He, S. Debray, and G. Andrews. Compressing dynamic data structures in operating system kernels. In *Proc. Workshop on Optimizations for DSP and Embedded Systems (ODES-7)*, Mar. 2009. 14

[45] H. He, S. K. Debray, and G. R. Andrews. The revenge of the overlay: automatic compaction of os kernel code via on-demand code loading. In *EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software*, pages 75–83, 2007. 14

[46] H. He, J. Trimble, S. Perianayagam, S. Debray, and G. Andrews. Code compaction of an operating system kernel. In *Proc. Fifth International Symposium on Code Generation and Optimization (CGO)*, pages 283–295, Mar. 2007. 14

[47] K. Heffner and C. S. Collberg. The obfuscation executive. In *Information Security, 7th International Conference*, pages 428–440. Springer Verlag, 2004. Lecture Notes in Computer Science, #3225. 14

[48] G. Hoglund and G. McGraw. *Exploiting Online Games: Cheating Massively Distributed Systems*. Addison-Wesley, 2007. 5

[49] B. Horne, L. Matheson, C. Sheehan, and R. E. Tarjan. Dynamic self-checking techniques for improved tamper resistance. In *Security and Privacy in Digital Rights Management, ACM CCS-8 Workshop DRM 2001*, Philadelphia, PA, USA, Nov. 2001. Springer Verlag, LNCS 2320. 5, 8

[50] M. Jacob, M. H. Jakubowski, and R. Venkatesan. Towards integral binary execution: implementing oblivious

hashing using overlapped instruction encodings. In *Proceedings of the 9th workshop on Multimedia & security*, pages 129–140, New York, NY, USA, 2007. ACM Press. 8

[51] H. Kim and J. Mayer. Data hiding for binary documents robust to print-scan, photocopy and geometric distortions. In *Computer Graphics and Image Processing, 2007. SIBGRAPI 2007. XX Brazilian Symposium on*, pages 105–112. IEEE, 2007. 4

[52] N. Krishnamoorthy, S. K. Debray, and K. Fligg. Static detection of disassembly errors. In *Proc. 16th. IEEE Working Conference on Reverse Engineering (WCRE)*, pages 259–268, Oct. 2009. 14

[53] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004. 15

[54] C. Linn and S. Debray. Obfuscation of executable code to improve resistance to static disassembly. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 290–299, 2003. 14

[55] C. Linn, S. Debray, and J. Kececioglu. Enhancing software tamper-resistance via stealthy address computations. In *19th Computer Security Applications Conference*, 2003. 14

[56] C. Linn, M. Rajagopalan, S. Baker, C. Collberg, S. Debray, and J. Hartman. Protecting against unexpected system calls. In *Proc. Usenix Security '05*, pages 239–254, Aug. 2005. 14

[57] J. Liu and S. Ram. PROMS: A system for harvesting and managing data provenance. In *Proceedings of 16th Workshop in Information Technology and System*, 2006. 14

[58] R. Lu, X. Lin, X. Liang, and X. Shen. Secure provenance: the essential of bread and butter of data forensics in cloud computing. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 282–292. ACM, 2010. 10

[59] M. Madou, B. Anckaert, S. K. Debray, B. De Sutter, and K. De Bosschere. Software protection through dynamic code mutation. In *Proc. Sixth International Workshop on Information Security Applications*, Aug. 2005. 14

[60] L. Moreau, J. Freire, J. Futrelle, R. E. McGrath, J. Myers, and P. Paulson. The open provenance model: An overview. In J. Freire, D. Koop, and L. Moreau, editors, *IPAW*, volume 5272 of *Lecture Notes in Computer Science*, pages 323–326. Springer, 2008. 5

[61] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer. Provenance-aware storage systems. In *USENIX Annual Technical Conference, General Track*, pages 43–56. USENIX, 2006. 4, 11

[62] G. Myles. *Software Theft Detection Through Program Identification*. PhD thesis, University of Arizona, 2006. 14

[63] G. Myles and C. Collberg. Software watermarking through register allocation: Implementation, analysis, and attacks. In *ICISC'2003 (International Conference on Information Security and Cryptology)*, 2003. 14

[64] G. Myles and C. Collberg. Detecting software theft via whole program path birthmarks. In *Information Security, 7th International Conference*, 2004. 14

[65] G. Myles and C. Collberg. k-gram based software birthmarks. In *Proceedings of SAC*, 2005. 5, 14

[66] G. Myles, C. Collberg, Z. Heidepriem, and A. Navabi. The evaluation of two software watermarking algorithms. *Software: Practice and Experience*, 35(10):923–938, 2005. 14

[67] T. Oinn, M. Addis, J. Ferris, D. Marvin, T. Carver, M. R. Pocock, and A. Wipat. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20, 2004. 5

[68] S. Perinayagam, H. He, M. Rajagopalan, G. Andrews, and S. Debray. Profile-guided specialization of an operating system kernel. In *Proc. Workshop on Binary Instrumentation and Applications*, Oct. 2006. 14

[69] J. Pollet. Electricity for free? the dirty underbelly of scada and smart meters, 2010. `http://media.blackhat.com/bh-us-10/whitepapers/Pollet_Cummins/BlackHat-USA-2010-Pollet-Cummings-RTS-Electricity-for-Free-wp.pdf`. 5

[70] I. Popov, S. K. Debray, and G. R. Andrews. Binary obfuscation using signals. In *Proc. Usenix Security 2007*, pages 275–290, Aug. 2007. 14

[71] M. Rajagopalan. *Optimizing system performance and dependability using compiler techniques*. PhD thesis, Dept. of Computer Science, University of Arizona, 2006. 14

[72] M. Rajagopalan, M. Hiltunen, T. Jim, and R. Schlichting. Authenticated system calls. In *Proc. IEEE International Symposium on Dependable Systems and Networks (DSN)*, June 2005. 14

[73] M. Rajagopalan, S. Perinayagam, H. He, G. Andrews, and S. Debray. Binary rewriting of an operating system kernel. In *Proc. Workshop on Binary Instrumentation and Applications*, Oct. 2006. 14

[74] S. Ram and J. Liu. Understanding the semantics of data provenance to support active conceptual modeling. In *Active Conceptual Modeling of Learning*, LNCS, pages 17–29, 2006. 14

[75] S. Ram and J. Liu. A semiotics framework for analyzing data provenance research. *Journal of Computing Science and Engineering*, 2(3):221–248, 2008. 14

[76] S. Ram and J. Liu. A new perspective on semantics of data provenance. In *Proceedings of the First International Workshop on the role of Semantic Web in Provenance Management (SWPM 2009)*, 2009. 1, 14

[77] S. Ram and J. Liu. Provenance management in biosicences. In *Lecture Notes in Computer Science 6413*, pages 54–64, 2010. 14

[78] S. Rozsnyai, A. Slominski, and Y. Doganata. Large-scale distributed storage system for business provenance. Technical report, 2011. 5

[79] S. E. Siwek. Video games in the 21st century: The 2010 report, 2010. `http://www.theesa.com/facts/pdfs/VideoGames21stCentury_2010.pdf`. 5

[80] J. Smith and C. Hudson. Inside virtual goods: The us virtual goods market 2010–2011, 2010. `http://www.insidevirtualgoods.com/us-virtual-goods`. 5

[81] S. K. Udupa, S. K. Debray, and M. Madou. Deobfuscation: Reverse engineering obfuscated code. In *WCRE '05: Proceedings of the 12th Working Conference on Reverse Engineering*, pages 45–54, Washington, DC, USA, 2005. IEEE Computer Society. 14

[82] A. Varna, S. Rane, and A. Vetro. Data hiding in hard-copy text documents robust to print, scan and photocopy operations. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 1397–1400. IEEE, 2009. 4

[83] R. Villn, S. Voloshynovskiy, O. Koval, F. Deguillaume, and T. Pun. Tamper-proofing of electronic and printed text documents via robust hashing and data-hiding. pages 65051T–65051T–12, 2007. 4

[84] R. Villn, S. Voloshynovskiy, O. Koval, J. Vila, E. Topak, F. Deguillaume, Y. Rytsar, and T. Pun. Text data-hiding for digital and printed documents: Theoretical and practical considerations. In *In Proceedings of SPIE-IS&T Electronic Imaging 2006, Security, Steganography, and Watermarking of Multimedia Contents VIII*, 2006. 4

[85] G. Wurster, P. van Oorschot, and A. Somayaji. A generic attack on checksumming-based software tamper resistance. In *2005 IEEE Symposium on Security and Privacy*, pages 127–138, 2005. 8

[86] J. Yan and B. Randell. A systematic classification of cheating in online games. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Networkand system support for games*, pages 1–9, New York, NY, USA, 2005. ACM. 5