Improving Data Reliability Through Assured Digital Signing

[1]BOMMANA SIVA KUMAR, [2]REPALLI LAVANYA LATHA
[1]Senior Software Engineer, [2]Advisor
[1]Hyderabad, Telangana, 500079, India[2]

ABSTRACT

Digital signatures are an important mechanism for ensuring data trustworthiness via source authenticity, integrity, and source non-repudiation. However, their trustworthiness guarantee can be subverted in the real world by sophisticated attacks, which can obtain cryptographically legitimate digital signatures without actually compromising the private signing key. This problem cannot be adequately addressed by a purely cryptographic approach, by the revocation mechanism of Public Key Infrastructure (PKI) because it may take a long time to detect the compromise, or by using tamper-resistant hardware because the attacker does not need to compromise the hardware. This problem will become increasingly more important and evident because of stealthy malware (or Advanced Persistent Threats).

In this paper, we propose a novel solution, dubbed Assured Digital Signing (ADS), to enhancing the data trustworthiness vouched by digital signatures. In order to minimize the modifications to the Trusted Computing Base (TCB), ADS simultaneously takes advantage of trusted computing and virtualization technologies. Specifically, ADS allows a signature verifier to examine not only a signature's cryptographic validity but also its system security validity that the private signing key and the signing function are secure, despite the powerful attack that the signing application program and the general-purpose Operating System (OS) kernel are malicious. The modular design of ADS makes it application-transparent (i.e., no need to modify the application source code in order to deploy it) and almost hypervisor-independent (i.e., it can be implemented with any Type I hypervisor). To demonstrate the feasibility of ADS, we report the implementation and analysis of an Xen-based ADS system.

1. INTRODUCTION

DIGITAL signatures are an important tool for ensuring data trustworthiness. The cryptographic assurance of digital signatures is well understood [1], assuming the private signing keys are not compromised (despite side channel attacks [2]). An appreciated problem is to attain a stronger signature trustworthiness than the cryptographic assurance.

However, existing solutions to this problem are not sufficient. Specifically, the cryptographic approach—including digital signatures of various flavors: threshold signature [3], proactive signatures [4], forward-secure signature [5], [6], key-insulated signature [7], and intrusion- resilient signatures [8]—can mitigate, but cannot prevent, the compromise of signature trustworthiness.

PKI-like key revocation mechanisms are not sufficient because the compromise may not be detected until after a long time. It is also not sufficient to put the private signing keys in tamper resistant hardware devices [9]. This is because the attacker can compromise the signing functions without compromising the private signing keys and without compromising the hardware devices.

For example, the attacker uses stealthy malware to penetrate into the Operating System (OS) kernel and then asks the device to sign the attacker's messages [10], [11].

In order to enhance signature trustworthiness in the real world, we need to address such powerful attacks.

2. LITERATURE SURVEY

a. A Framework For Digital Signature

Digital signatures allow you to protect the integrity of a message. They are used to verify that a message sent was sent by the actual user that sent the message and was modified in transit. Most web apps handle message integrity by using TLS, like HTTPS, to secure the connection between the client and server. Sometimes though, we have representations that are going to be forwarded to more than one recipient. Some representations may hop around from server to server. In this case, TLS is not enough. There needs to be a mechanism to verify who sent the original representation and that they actually sent that message. This is where digital signatures come in.

Resteasy has developed its own protocol for digital signatures. It allows you to attach one or more digital signatures to a request or response via a Content-Signature header. Signatures are comprised of the message body and an arbitrary set of metadata. Adding metadata to the signature calculation gives you a lot of flexibility to piggyback various features like expiration and authorization.

A complete description of the protocol is available on Bill Burke's Blog. We are also submitting this protocol as an Internet-Draft at IETF to hopefully get it standardized in some way or form. There's also another blog about various uses you might find for the protocol. The rest easy distribution also has an example you can check out.

2.1.1 Signing API

To sign a request or response using the Resteasy client or server framework you need to create an instance of org.jboss.resteasy.security.signing.ContentSignatures. Content Signatures hold one or more signatures you want to attach to the request or response and models the Content-Signature header. You instantiate the Content Signatures object and then set the "Content-Signature" header of the request or response.

To sign a message you need a Private Key. This can be generated by Key Tool or manually using regular, standard JDK Signature APIs. The default algorithm used is an SHA1 256 hash with RSA, "SHA256withRSA". Resteasy supports any algorithm that uses private/public key pairs and available through the java.security.Signature class. The Content Signature class also allows you to add and control how various pieces of metadata are added to the Content-Signature header and the signature calculation.

If you are including more than one signature, then use the Content Signatures class to hold multiple signatures.

2.1.2. Using a KeyRepository

In the above example, you are managing your own keys. Resteasy can manage keys for you through a org.jboss.resteasy.security.KeyRepository. Currently, only Java Key Stores are supported, but you can implement and plug in your own implementation of this interface if you need a different mechanism to manage keys.

If you're going to use the built in Java KeyStore support, use Java keytool to generate a private and public key.

You can reference a Java key store you want the Resteasy signature framework to use within web.xml using either resteasy.keystore.classpath or resteasy.keystore.filename context parameters. You must also specify the password (sorry its clear text) using the resteasy.keystore.password context parameter.

You invoke the ContentSignature.setKeyAlias() method, resteasy will use a configured Key Repository to lookup that key alias within the repository. Alternatively, if you do not set the key alias via setKeyAlias(), Resteasy will use the signer attribute of the Content Signature which is set via the ContentSignature.setSigner() method.

You can also manually register your own instance of a KeyRepository within an Application class.

Client side, you can load a KeyStore manually, by instantiating an instance of org.jboss.resteasy.security.keys.KeyStoreKeyRepository. You then set a request attribute, "org.jboss.resteasy.security.keys.KeyRepository", with the value of the created instance. Use the ClientRequest.getAttributes() method to do this.

2.1.3. Signed annotation

An alternative to the manual way of signing using a ContentSignatures instance is to use the @org.jboss.resteasy.annotations.security.signature.Sign annotation. It is required that you configure a KeyRepository as described earlier.

This annotation also works with the client proxy framework.

2.1.4. Annotation-based verification

The easiest way to verify a signature sent in a HTTP request on the server side is to use the @@org.jboss.resteasy.annotations.security.signature.Verify (or @Verifications which is used to verify multiple signatures).

By default, Resteasy will look for a signer attribute in every contained signature to determine which key Alias to use to obtain a public key and verify the signatures. You can also specify which specific signatures you want to verify as well as define multiple verifications you want to happen via the @Verifications annotation.

Failed verifications will throw an org.jboss.resteasy.security.signing. UnauthorizedSignatureException. This causes a 401 error code to be sent back to the client. If you catch this exception using an Exception Handler you can browse the failure results.

2.1.5. Manual verification

If you want fine grain control over verification, this is an API to verify signatures manually. Its a little tricky because you'll need the raw bytes of the HTTP message body in order to verify the signature. You can get at an unmarshalled message body as well as the underlying raw bytes by using a org.jboss.resteasy.spi.MarshalledEntity injection.

MarshalledEntity is a generic interface. The template parameter should be the Java type you want the message body to be converted into. Use a ContentSignatures injection if you are expecting more than one signature within the Content-Signature header.

2.2 Performance Analysis

Due to the high scope of the software, the performance requirements are high. The speed at which the software is required to operate is nominal. A processing rate of 5-10 seconds per query is acceptable.

i. **Error message design**

The design of error messages is an important part of the user interface design. As user is bound to commit some errors or other while designing a system the system should be designed to be helpful by providing the user with information regarding the error he/she has committed.

ii. **Error detection:**

Even though every effort is make to avoid the occurrence of errors, still a small portion of errors are always likely to occur, these type of errors can be discovered by using validations to check input data.

The system is designed to be a user friendly one. In other words, the system has been designed to communicate effectively with the user. The system has been designed with Button.

3. <u>SYSTEM ANALYSIS</u>

3.1 Existing System

An appreciated problem is to attain stronger signature trustworthiness than the cryptographic assurance. However, existing solutions to this problem are not sufficient. Specifically, the cryptographic approach—including digital signatures of various flavors: threshold signature, proactive signatures, forward-secure signature, key-insulated signature, and intrusion-resilient signatures-can mitigate, but cannot prevent, the compromise of signature trustworthiness.

PKI-like key revocation mechanisms are not sufficient because the compromise may not be detected until after a long time. It is also not sufficient to put the private signing keys in tamper resistant hardware devices. This is because the attacker can compromise the signing functions without compromising the private signing keys and without compromising the hardware devices.

For example, the attacker uses stealthy malware to penetrate into the Operating System (OS) kernel and then asks the device to sign the attacker's messages. In order to enhance signature trustworthiness in the real world, we need to address such powerful attacks.

3.2 Proposed System

We propose Improving Data Reliability Through Assured Digital Signing (ADS), which allows a signature verifier to examine not only digital signatures' cryptographic validity as in the current daily routine practice, but also their system security validity that the private signing keys and the signing functions are secure.

In particular, ADS deals with the powerful attacks that the signing application program itself may be malicious (e.g., a backdoor was embedded by its vendor or developer), and that the underlying general-purpose OS kernel is malicious.

In order to minimize the modifications to the Trusted Computing Base (TCB), we propose a modular design of ADS, which simultaneously takes advantage of trusted computing and virtualization technologies.

4. FEASIBILITY STUDY

4.1 Introduction:

A feasibility analysis involves a detailed assessment of the need, value and practicality of a p systems development... Feasibility analysis n forms the transparent decisions at crucial points during the developmental process as we determine whether it is operationally, economically and technically realistic to proceed with a particular course of action.

Feasibility analysis can be used in each of the steps to assess the financial, technical and operational capacity to proceed with particular activities.

4.2 Types Of Feasibility:

A feasibility analysis usually involves a thorough assessment of the financial (value), technical (practicality), and operational (need) aspects of a proposal. In systems development projects, business managers are primarily responsible for assessing the operational feasibility of the system, and information technology (IT) analysts are responsible for assessing technical feasibility. Both then work together to prepare a cost–benefit analysis of the proposed system to determine its economic feasibility.

4.3 Operational Feasibility:

A systems development project is likely to be operationally feasible if it meets the 'needs' and expectations of the organization. User acceptance is an important determinant of operational feasibility. It requires careful consideration of:

- Corporate culture;
- Staff resistance or receptivity to change;
- Management support for the new system;
- The nature and level of user involvement in the development and implementation of the system; direct and indirect impacts of the new system on work practices;
- Anticipated performance and outcomes of the new system compared with the existing system;
- Training requirements and other change management strategies; and
- 'Pay back' periods (i.e., trade-off between long-term organizational benefits and short-term inefficiencies during system development and implementation).

4.4 Technical feasibility:

A systems development project may be regarded as technically feasible or practical if the organization has the necessary expertise and infrastructure to develop, install, operate and maintain the proposed system. Organizations will need to make this assessment based on:

- Knowledge of current and emerging technological solutions
- Availability of technically qualified staff in-house for the duration of the project and subsequent maintenance phase;
- Availability of infrastructure in-house to support the development and maintenance of the proposed system;
- Where necessary, the financial and/or technical capacity to procure appropriate infrastructure and expertise from outside;

- Capacity of the proposed system to accommodate increasing levels of use over the medium term;

The capacity of the proposed system to meet initial performance expectations and accommodate new functionality over the medium term.

4.5 Economical Feasibility:

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

4.6 Technical Feasibility:

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement; as only minimal or null changes are required for implementing this system.

4.7 Social Feasibility:

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

5. SYSTEM REQUIREMENTS

5.1 Hardware Requirements:

System	:	Pentium IV
Hard Disk	:	80 GB
RAM	:	512 MB

5.2 Software Requirements:

Operating System	:	Windows XP
Language	:	Java
DataBase	:	Mysql

5.3 Functional and Non-Functional Requirements:

5.3.1 Functional Requirements:

Inputs:

The inputs to the system will be manually fed by the One of the user. The inputs will be as follows.

i. **Username and Password:**

The user has to username and password as inputs for login.

ii. **Text:**
The user should give Text as a input means if the user wants to send any data to the another user then that data should be given to the system.

iii. **Receiver Data:**
The user should provide Receiver name which he wants to send the data. In this situation The receiver name should be the destination.

a. **Processing:**
The input data is processed by the model

Retrieving Data:
To Retrieve Data the Receiver should login into his account and then he need go to inbox there he need to enter the username and sign this username and sign will be generated by system itself these details are sender details by giving sender username and sign only the user can retrieve the data which was sent by the sender.

OUTPUT: Users has to register with login. Admin stores the all the details of users. Users can send mails to each other if the mails modified by the another user it will be displayed.

b. **Performance requirements:**
Due to the high scope of the software, the performance requirements are high. The speed at which the software is required to operate is nominal. A processing rate of 5-10 seconds per query is acceptable.

Error message design:
The design of error messages is an important part of the user interface design. As user is bound to commit some errors or other while designing a system the system should be designed to be helpful by providing the user with information regarding the error he/she has committed.

Error detection:
Even though every effort is make to avoid the occurrence of errors , still a small portion of errors are always likely to occur , these type of errors can be discovered by using validations to check input data.

The system is designed to be a user friendly one. In other words, the system has been designed to communicate effectively with the user. The system has been designed with Button.

5.3.2 Non Functional Requirements

Performance is measured in terms of the output provided by the application.

Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of users of the existing system to give the requirement specifications because they are the people who finally use the system.

The requirement specification for any system can be broadly stated as given below:

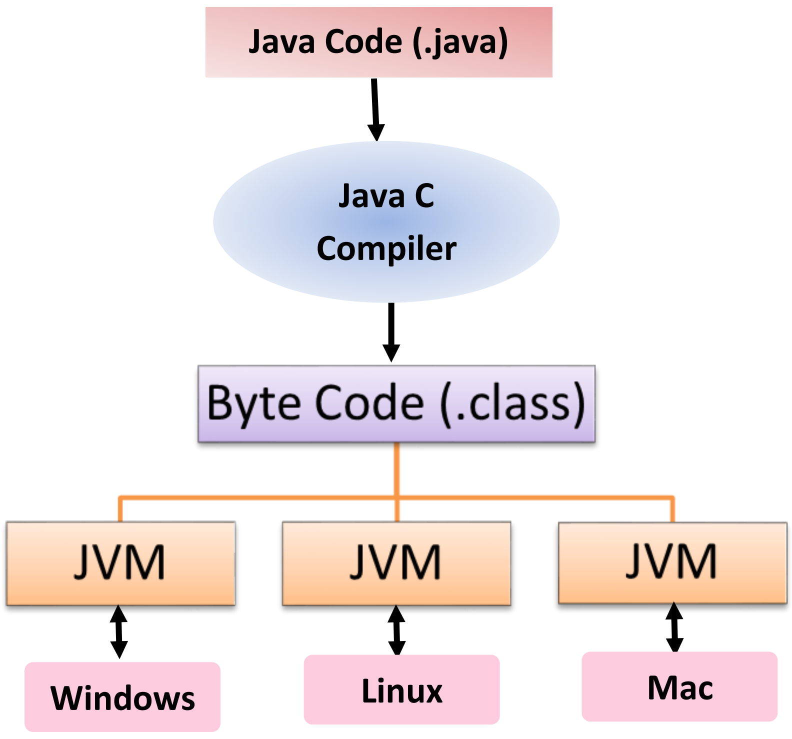
- The system should be able to interface with the existing system.
- The system should be accurate.

- The system should be better than existing system.
    - **Portability:** It should run on specified platforms successfully. To achieve this, we should test the product on all platforms before launching the product. If our project runs successfully on different platforms, then our system is portable in nature.
    - **Reliability:** The system should perform its intended functions under specified conditions. If our system satisfies all the specified conditions, then it is Reliable in nature.
    - **Reusability:** The system should be extremely reusable as a whole or part. Make the system modularize and make sure that modules are loosely coupled. This project is having reusability nature because we can reuse whole or part of this project on other systems.
    - **Robustness:** The system on the whole should be robust enough to perform well under different circumstances without any inconsistencies.
    - **Testability:** The product of a given development phase should satisfy the conditions imposed at the start of that phase.
    - **Usability:** It should be perfect and comfortable for users to work.
    - **Security:** The system is completely based on the security. This system will provide security base on the password.

6. SOFTWARE ENVIRONMENT

6.1 Introduction To Java:

Java technology is both a programming language and a platform.



The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple & Architecture Neutral
- Object oriented, Portable, Distributed & High Performance
- Interpreted, Multithreaded, Robust, Dynamic & Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.

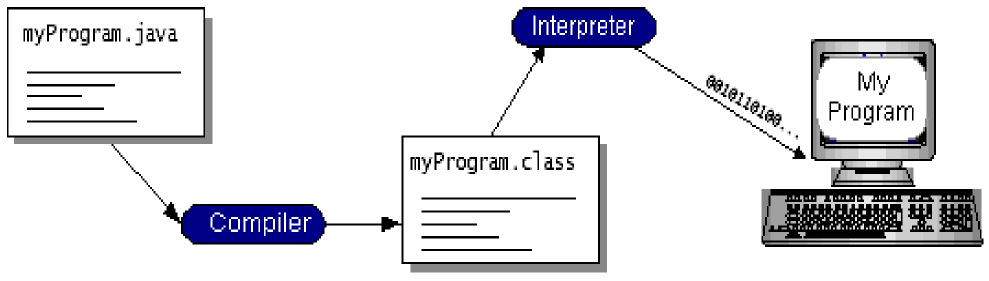
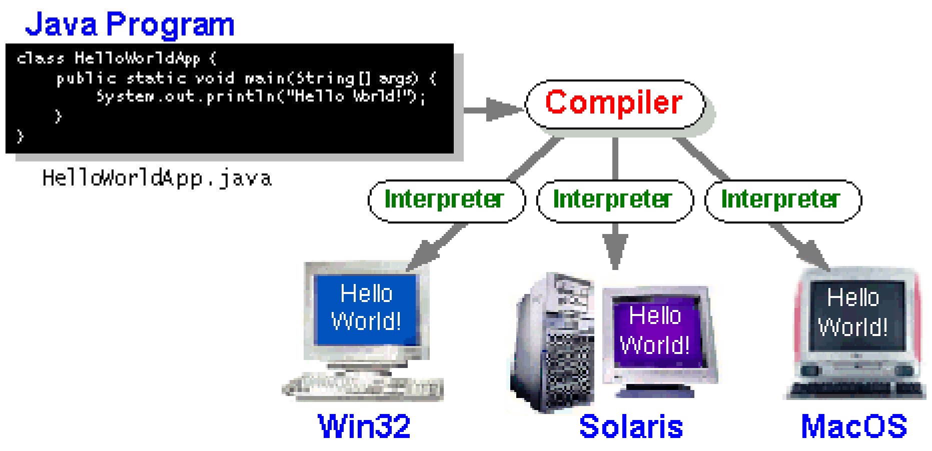


Fig: 6.1.1 WORKING OF JAVA PROGRAM

You can think of Java byte codes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make "write once, run anywhere" possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Fig 6.1.2 JAVA INTERPRETER

The Java Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components:

The *Java Virtual Machine* (Java VM)

The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.

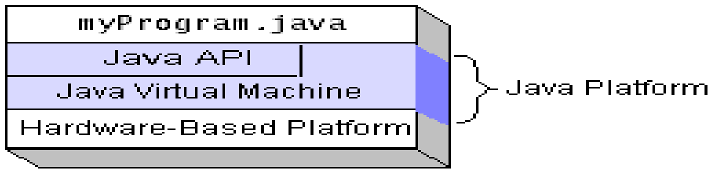


Fig 6.1.3 RUNNING OF JAVA

Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

What Can Java Technology Do?

The most common types of programs written in the Java programming language are *applets* and *applications*. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a *servlet*. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- **The essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets**: The set of conventions used by applets.

- **Networking**: URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.
- **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security**: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- **Software components**: Known as JavaBeans™, can plug into existing component architectures.
- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC™)**: Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.

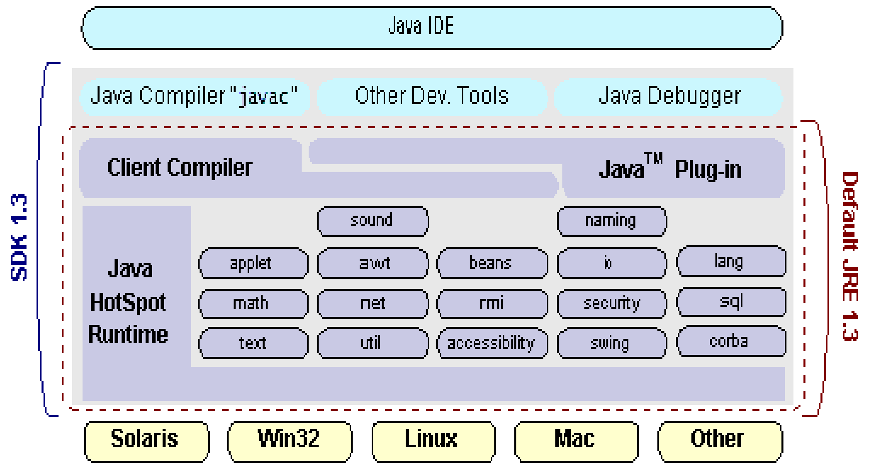


Fig 6.1.4 JAVA 2 SDK ARCHITECTURE

How Will Java Technology Change My Life?

We can't promise you fame, fortune, or even a job if you learn the Java programming language. Still, it is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

- **Get started quickly**: Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code**: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
- **Write better code**: The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.
- **Develop programs more quickly**: Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.
- **Avoid platform dependencies with 100% Pure Java**: You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure Java™ Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.

- **Write once, run anywhere**: Because 100% Pure Java programs are compiled into machine-independent byte codes, they run consistently on any Java platform.
- **Distribute software more easily**: You can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded "on the fly," without recompiling the entire program.

6.2 Swing:

Introduction To Swing:

Swing contains all the components. It's a big library, but it's designed to have appropriate complexity for the task at hand – if something is simple, you don't have to write much code but as you try to do more your code becomes increasingly complex. This means an easy entry point, but you've got the power if you need it.

Swing has great depth. This section does not attempt to be comprehensive, but instead introduces the power and simplicity of Swing to get you started using the library. Please be aware that what you see here is intended to be simple. If you need to do more, then Swing can probably give you what you want if you're willing to do the research by hunting through the online documentation from Sun.

Benefits Of Swing:

Swing components are Beans, so they can be used in any development environment that supports Beans. Swing provides a full set of UI components. For speed, all the components are lightweight and Swing is written entirely in Java for portability.

Swing could be called "orthogonality of use;" that is, once you pick up the general ideas about the library you can apply them everywhere. Primarily because of the Beans naming conventions.

Keyboard navigation is automatic – you can use a Swing application without the mouse, but you don't have to do any extra programming. Scrolling support is effortless – you simply wrap your component in a JScrollPane as you add it to your form. Other features such as tool tips typically require a single line of code to implement.

Swing also supports something called "pluggable look and feel," which means that the appearance of the UI can be dynamically changed to suit the expectations of users working under different platforms and operating systems. It's even possible to invent your own look and feel.

Domain Description:

Data mining involves the use of sophisticated data analysis tools to discover previously unknown, valid patterns and relationships in large data sets. These tools can include statistical models, mathematical algorithms, and machine learning methods (algorithms that improve their performance automatically through experience, such as neural networks or decision trees). Consequently, data mining consists of more than collecting and managing data, it also includes analysis and prediction.

Data mining can be performed on data represented in quantitative, textual, or multimedia forms. Data mining applications can use a variety of parameters to examine the data. They include association (patterns where one event is connected to another event, such as purchasing a pen and purchasing paper), sequence or path analysis (patterns where one event leads to another event, such as the birth of a child and purchasing diapers), classification (identification of new patterns, such as coincidences between duct tape purchases and plastic sheeting purchases), clustering (finding and visually documenting groups of previously unknown facts, such as geographic location and brand preferences), and forecasting (discovering patterns from which one can make reasonable predictions regarding future activities, such as the prediction that people who join an athletic club may take exercise classes)

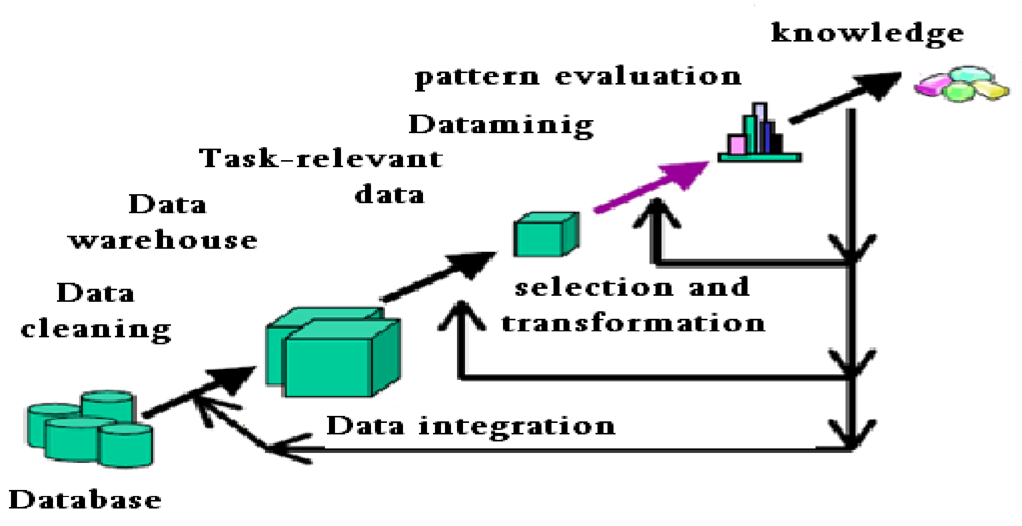


Figure 6.2.1: knowledge discovery process

Data Mining Uses:
Data mining is used for a variety of purposes in both the private and public sectors.

- Industries such as banking, insurance, medicine, and retailing commonly use data mining to reduce costs, enhance research, and increase sales. For example, the insurance and banking industries use data mining applications to detect fraud and assist in risk assessment (e.g., credit scoring).

7. SYSTEM DESIGN

UML Diagrams:
Description
Object Oriented Analysis

An **object-oriented system** is composed of objects. The behavior of the system is achieved through collaboration between these objects, and the state of the system is the combined state of all the objects in it. Collaboration between objects involves them sending messages to each other.

Object Oriented Analysis aims to model the problem domain, the problem we want to solve by developing an object-oriented (OO) System. The source of the analysis is a written requirement statement, and/or written use cases, UML diagrams can be used to illustrate the statements.

An analysis model will not take into account implementation constraints, such as concurrency, distribution, persistence, or inheritance, nor how the system will be built The model of a system can be divided into multiple domains each of which are separately analyzed, and represent separate business, technological, or conceptual areas of interest The result of object-oriented analysis is a description of what is to be built, using concepts and relationships between concepts, often expressed as a conceptual model.

Any other documentation that is needed to describe what is to be built is also included in the result of the analysis. That can include a detailed user interface mock-up document.

Object Oriented Design

Object-Oriented Design (**OOD**) is an activity where the designers are looking for logical solutions to solve a problem, using Objects Object-oriented design takes the conceptual model that is the result of object-oriented analysis, and adds implementation constraints imposed by the environment, the programming language and the chosen tools, as well as architectural assumptions chosen as basis of Design.

The concepts in the conceptual model are mapped to concrete classes, to abstract interfaces in APIs and to roles that the objects take in various situations. The interfaces and their implementations for stable concepts can be made available as reusable services.

Concepts identified as unstable in object-oriented analysis will form basis for policy classes that make decisions, implement environment-specific or situation specific logic or algorithms.

The result of the object-oriented design is a detail description how the system can be built; using objects. Object-oriented software engineering (OOSE) is an object modeling language and Methodology.

OOSE was developed by Ivar Jacobson in 1992 while at Objectory AB. It is the first object-oriented design methodology to employ use cases to drive software design. It also uses other design products similar to those used by OMT.

The tool Objector was created by the team at Objectory AB to implement the OOSE methodology. After success in the marketplace, other tool vendors also supported OOSE after Rational bought Objectory AB, the OOSE notation, methodology, and tools became superseded.

As one of the primary sources of the Unified Modeling Language (UML), concepts and notation from OOSE have been incorporated into UML. The methodology part of OOSE has since evolved into the Rational Unified Process (RUP)The OOSE tools have been replaced by tools supporting UML and RUPOOSE has been largely replaced by the UML notation and by the RUP methodology

Unified Modeling Language

The heart of object-oriented problem solving is the construction of a model. The model abstracts the essential details of the underlying problem from its usually complicated real world. Several modeling tools are wrapped under the heading of the

UML, which stands for Unified Modeling Language™. The purpose of this course is to present important highlights of the UML.

- Use case diagram
- Class diagram
- Object diagram
- Sequence diagram
- Collaboration diagram
- State chart diagram
- Activity diagram
- Component diagram
- Deployment diagram

Why is UML important?

Let's look at this question from the point of view of the construction trade. Architects design buildings. Builders use the designs to create buildings. The more complicated the building, the more critical the communication between architect and builder.

Actors

Actors are the users of the system being modeled. Each Actor will have a well-defined role, and in the context of that role have useful interactions with the system. A person may perform the role of more than one Actor, although they will only assume one role during one use case interaction. An Actor role may be performed by a non-human system, such as another computer program.

Use Cases

Use case Diagrams represent the functionality of the system from a user's point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view. Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer etc., or another system like central database.

Sequence Diagram

Class and object diagrams are static model views. Interaction diagrams are dynamic. They describe how objects collaborate.

A sequence diagram is an interaction diagram that details how operations are carried out, what messages are sent and when Sequence diagrams are organized according to time?

The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Class Diagram

A Class diagram gives an overview of a system by showing its classes and the relationships among them. Class diagrams are static -- they display what interacts but not what happens when they do interact.

Our class diagram has three kinds of relationships.

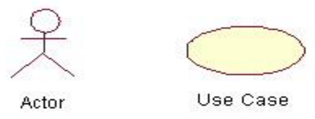
Association -- a relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other in order to perform its work. In a diagram, an association is a link connecting two classes.

Aggregation -- an association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole.

Generalization -- an inheritance link indicating one class is a super class of the other. A generalization has a triangle pointing to the super class.

7.1 Use-Case diagram:

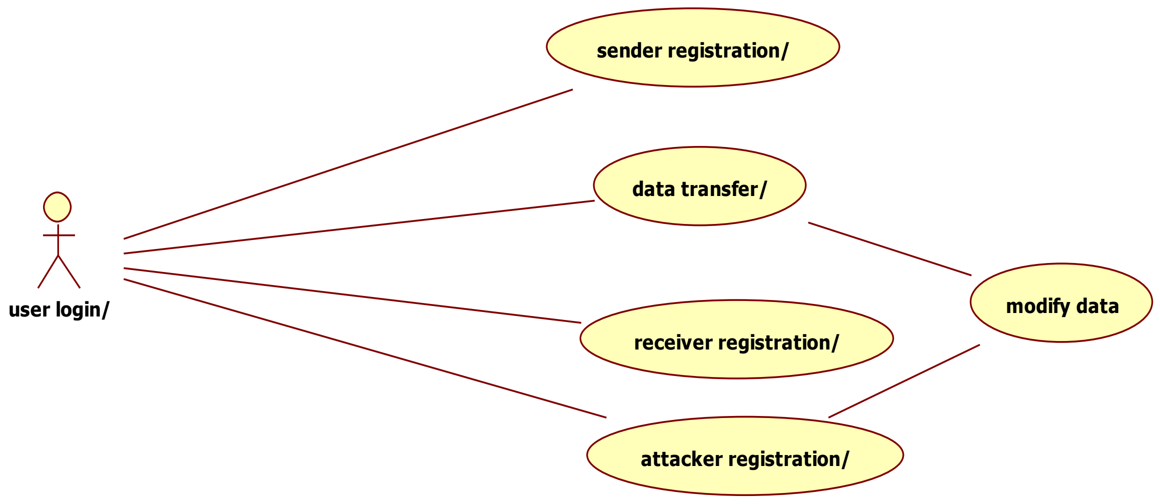
A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.



An actor is represents a user or another system that will interact with the system you are modeling. A use case is an external view of the system that represents some action the user might perform in order to complete a task.

Contents:

- Use cases, Actors
- Dependency, Generalization, and association relationships
- System boundary



7.2 Class Diagram:

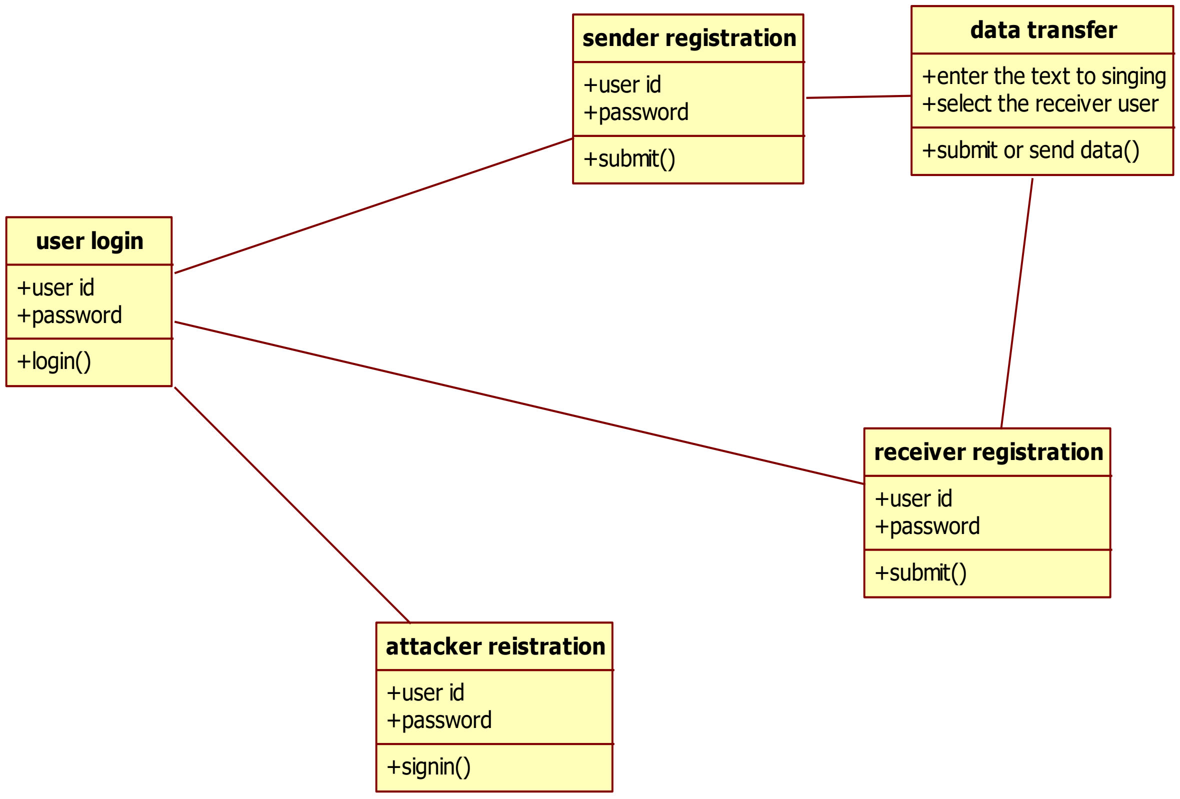
Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation.

These perspectives become evident as the diagram is created and help solidify the design. Class diagrams are arguably the most used UML diagram type. It is the main building block of any object oriented solution. It shows the classes in a system, attributes and operations of each class and the relationship between each class.

In most modeling tools a class has three parts, name at the top, attributes in the middle and operations or methods at the bottom. In large systems with many classes related classes are grouped together to to create class diagrams.

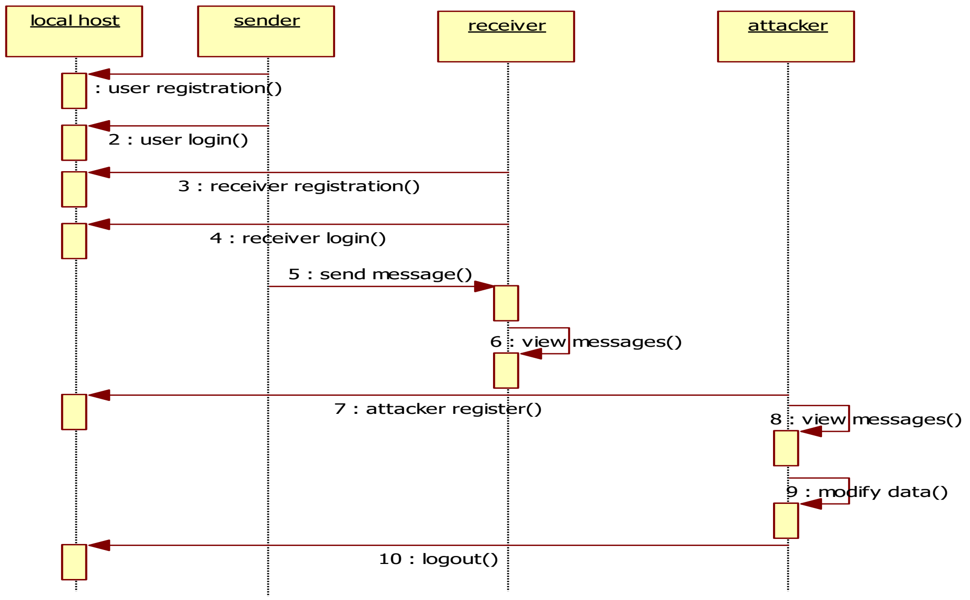
Different relationships between diagrams are show by different types of Arrows. Below is a image of a class diagram.

UML Class Diagram with Relationships



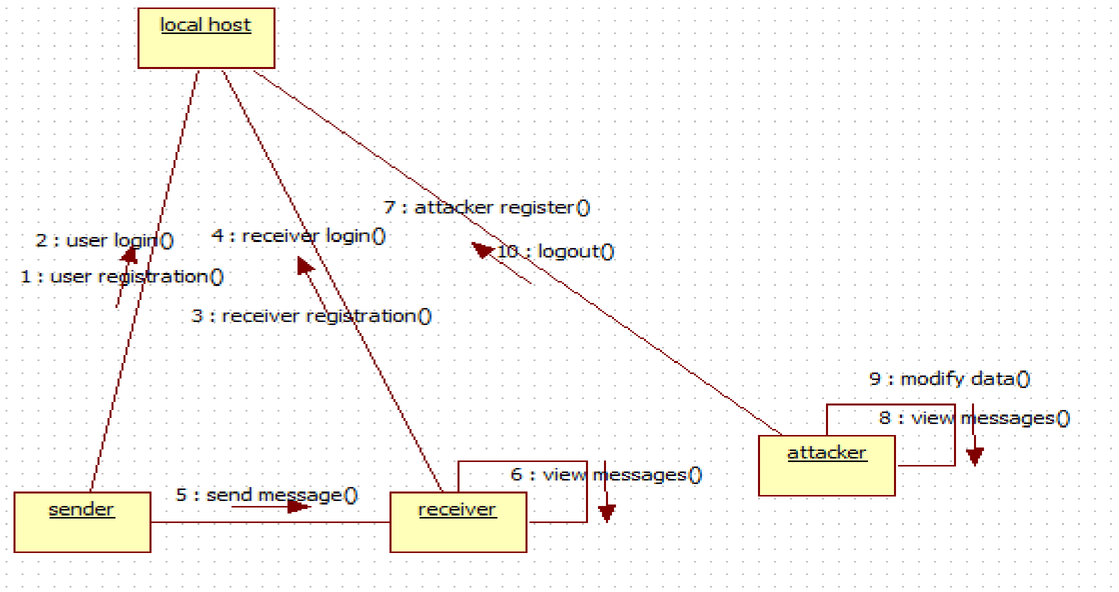
7.3 Sequence Diagram

Sequence diagrams in UML shows how object interact with each other and the order those interactions occur. It's important to note that they show the interactions for a particular scenario. The processes are represented vertically and interactions are show as arrows. This article explains the purpose and the basics of Sequence diagrams.



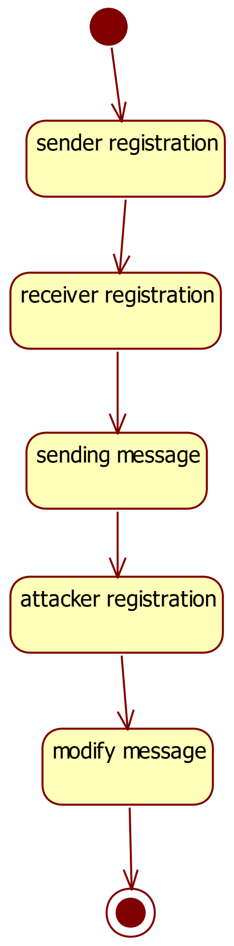
7.4 Collaboration diagram

Communication diagram was called collaboration diagram in UML 1. It is similar to sequence diagrams but the focus is on messages passed between objects. The same information can be represented using a sequence diagram and different objects. Click here to understand the differences using an example.



7.5 State machine diagrams

State machine diagrams are similar to activity diagrams although notations and usage changes a bit. They are sometime known as state diagrams or start chart diagrams as well. These are very useful to describe the behavior of objects that act different according to the state they are at the moment. Below State machine diagram show the basic states and actions.



State Machine diagram in UML, sometime referred to as State or State chart diagram

7.6 Activity diagram:

Activity diagrams describe the workflow behavior of a system. Activity diagrams are similar to state diagrams because activities are the state of doing something. The diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel.

How to Draw: Activity Diagrams

Activity diagrams show the flow of activities through the system. Diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities. A fork is used when multiple activities are occurring at the same time. The diagram below shows a fork after activity1. This indicates that both activity2 and activity3 are occurring at the same time. After activity2 there is a branch. The branch describes what activities will take place based on a set of conditions. All branches at some point are followed by a merge to indicate the end of the conditional behavior started by that branch. After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state. .

When to Use: Activity Diagrams

Activity diagrams should be used in conjunction with other modeling techniques such as interaction diagrams and state diagrams. The main reason to use activity diagrams is to model the workflow behind the system being designed. Activity Diagrams are also useful for: analyzing a use case by describing what actions need to take place and when they should occur; describing a complicated sequential algorithm; and modeling applications with parallel processes.

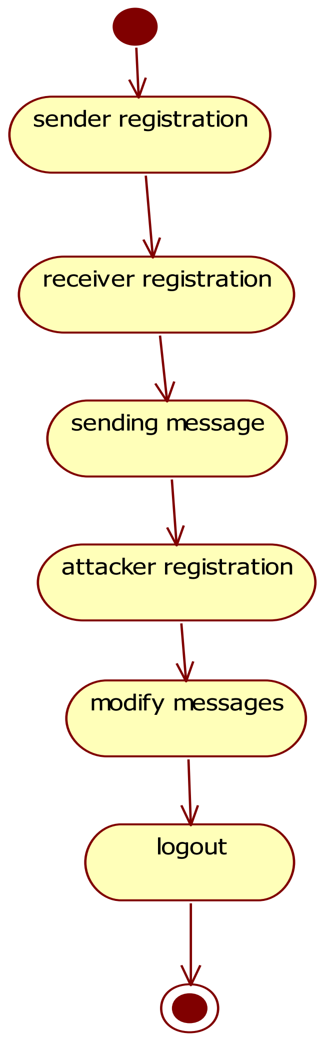


Fig. 7.6 Activity Diagram

7.7 Component diagram

A component diagram displays the structural relationship of components of a software system. These are mostly used when working with complex systems that has many components.

Components communicate with each other using interfaces. The interfaces are linked using connectors.

7.8 Deployment Diagram

A deployment diagrams shows the hardware of your system and the software in those hardware. Deployment diagrams are useful when your software solution is deployed across multiple machines with each having a unique configuration.

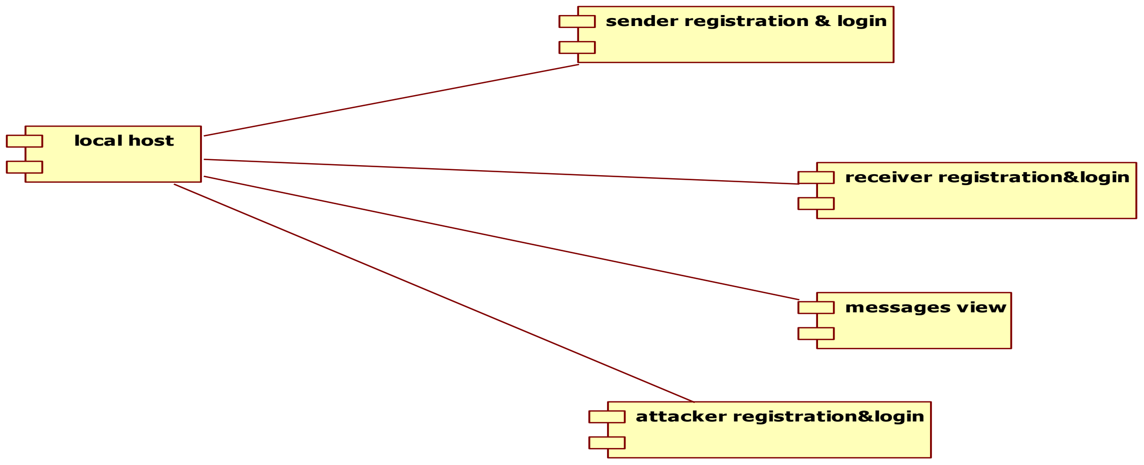

sender registration & login
local host
receiver registration&login
messages view
attacker registration&login

Fig. 7.7 Component diagram

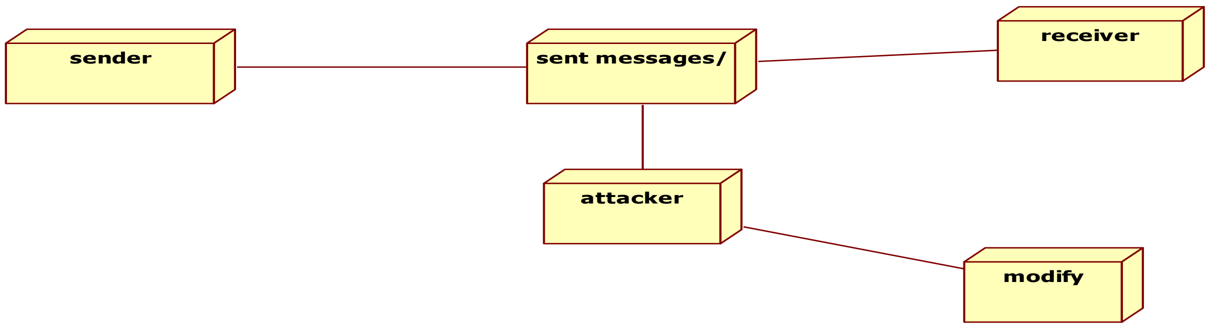


Fig. 7.8 Deployment Diagram

8. <u>IMPLEMENTATION</u>

8.1 Modules:

- Sender Login
- Public, Private key Generation
- Signature Creation
- Transferring
- Receiver Login
- Signature Verification

8.2 Module Description:

8.2.1 Sender Login:

This sender login consists of select file name, browse button, send button. If user want to send any file means he select any file from the browse button and he can click send button automatically the file which we selected will send to the receiver path.

8.2.2 Public, Private key Generation:

This Module Will Transfer The Data From Sender To Receiver**.**

8.2.3 Signature Creation:

In cryptography, a public key is a value provided by some designated authority as an encryption key that, combined with a private key derived from the public key, can be used to effectively encrypt messages and digital signatures. The use of combined public and private keys is known as asymmetric cryptography. A system for using public keys is called a public key infrastructure (PKI).

In cryptography, a private or secret key is an encryption/decryption key known only to the party or parties that exchange secret messages. In traditional secret key cryptography, a key would be shared by the communicators so that each could encrypt and decrypt messages.

The risk in this system is that if either party loses the key or it is stolen, the system is broken. A more recent alternative is to use a combination of public and private keys.

In this system, a public key is used together with a private key. See public key infrastructure (PKI) for more information.

8.2.4 Transferring:

Simpler signatures may be more effective for some companies and departments. but this only increases the need for consistency and composition strategy. Simpler signatures may be more effective for some companies and departments. but this only increases the need for consistency and composition strategy.

8.2.5 Receiver Login:

This receiver module will select the path where send file need to store and it also receive the data or files. It saves the received file.

8.2.6 Signature Verification:

Signature scheme is a mathematical scheme for demonstrating the authenticity of a digital message or document. A valid digital signature gives a recipient reason to believe that the message was created by a known sender such that they cannot deny sending it (authentication and non-repudiation) and that the message was not altered in transit (integrity).

Digital signatures are commonly used for software distribution, financial transactions, and in other cases where it is important to detect forgery or tampering.

8.3 PUBLIC KEY INFRASTRUCTURE (PKI)

A **PKI** (public key infrastructure) enables users of a basically unsecure public network such as the Internet to securely and privately exchange data and money through the use of a public and a private cryptographic key pair that is obtained and shared through a trusted authority. The public key infrastructure provides for a digital certificate that can identify an individual or an organization and directory services that can store and, when necessary, revoke the certificates. Although the components of a PKI are generally understood, a number of different vendor approaches and services are emerging. Meanwhile, an Internet standard for PKI is being worked on.

The public key infrastructure assumes the use of *public key cryptography*, which is the most common method on the Internet for authenticating a message sender or encrypting a message. Traditional cryptography has usually involved the creation and sharing of a secret key for the encryption and decryption of messages. This secret or private key system has the significant flaw that if the key is discovered or intercepted by someone else, messages can easily be decrypted. For this reason, public key cryptography and the public key infrastructure is the preferred approach on the Internet. (The private key system is sometimes known as *symmetric cryptography* and the public key system as *asymmetric cryptography*.)

A public key infrastructure consists of:

- A certificate authority (CA) that issues and verifies digital certificate. A certificate includes the public key or information about the public key
- A registration authority (RA) that acts as the verifier for the certificate authority before a digital certificate is issued to a requestor
- One or more directories where the certificates (with their public keys) are held
- A certificate management system

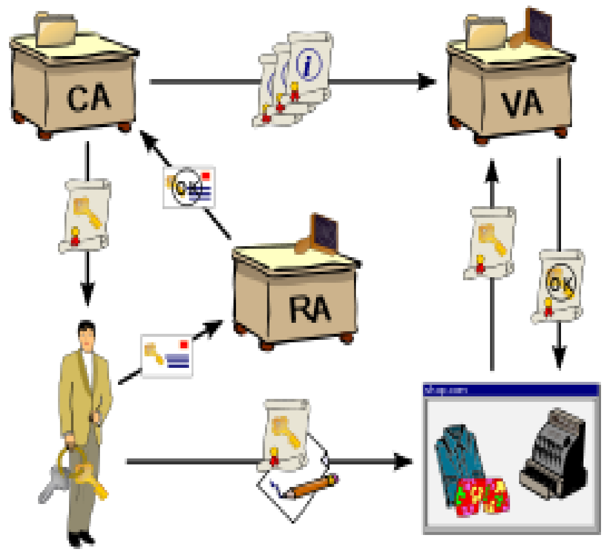


Fig 8.3.1 Public Key Infrastructure

How Public and Private Key Cryptography Works

In public key cryptography, a public and private key are created simultaneously using the same algorithm (a popular one is known as RSA) by a certificate authority (CA). The private key is given only to the requesting party and the public key is made publicly available (as part of a digital certificate) in a directory that all parties can access.

The private key is never shared with anyone or sent across the Internet. You use the private key to decrypt text that has been encrypted with your public key by someone else (who can find out what your public key is from a public directory). Thus, if I send you a message, I can find out your public key (but not your private key) from a central administrator and encrypt a message to you using your public key.

When you receive it, you decrypt it with your private key. In addition to encrypting messages (which ensures privacy), you can authenticate yourself to me (so I know that it is really you who sent the message) by using your private key to encrypt a digital certificate.

When I receive it, I can use your public key to decrypt it.

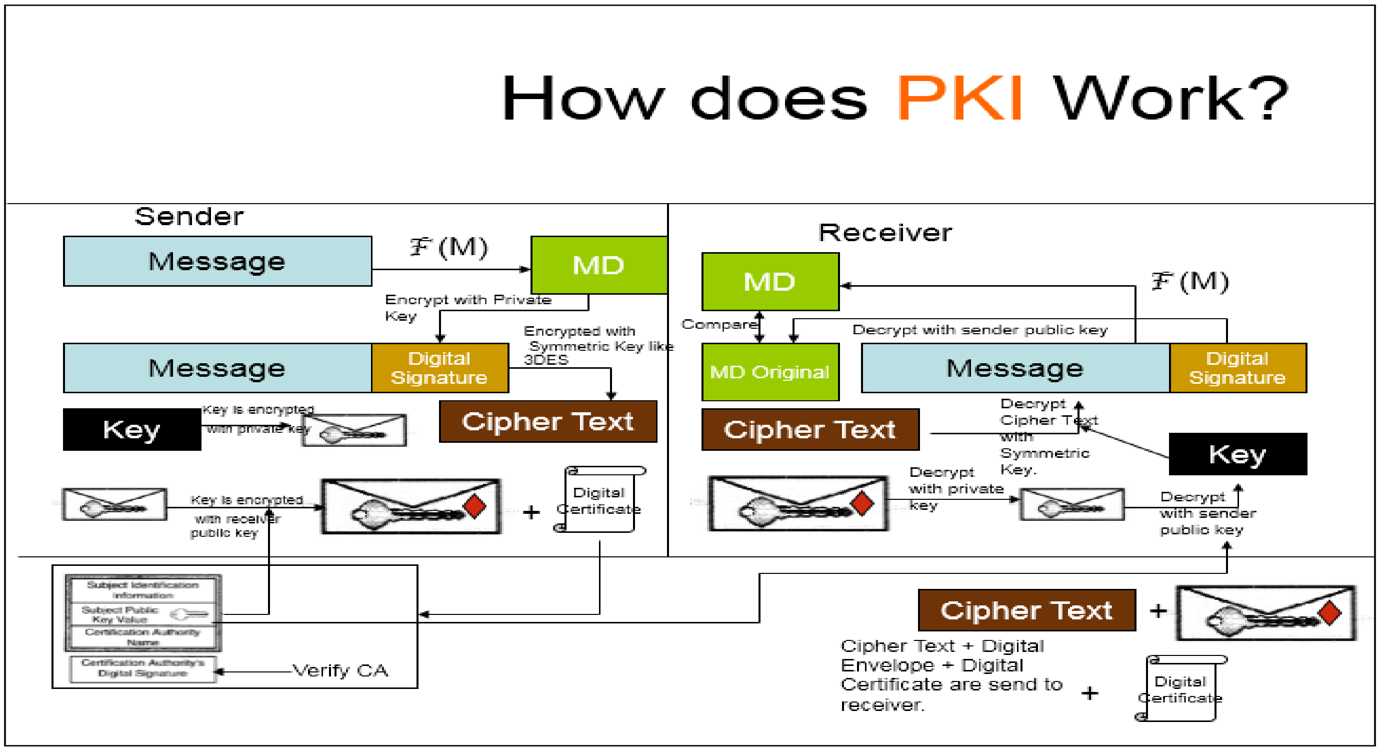

How does PKI Work?
Sender
Message
ℱ(M)
MD
Encrypt with Private Key
Encrypted with Symmetric Key like 3DES
Message
Digital Signature
Key
Key is encrypted with private key
Cipher Text
Key is encrypted with receiver public key
Digital Certificate
Receiver
MD
ℱ(M)
Compare
Decrypt with sender public key
MD Original
Message
Digital Signature
Cipher Text
Decrypt Cipher Text with Symmetric Key.
Key
Decrypt with private key
Decrypt with sender public key
Subject Identification Information
Subject Public Key Value
Certification Authority Name
Certification Authority's Digital Signature
Verify CA
Cipher Text
Cipher Text + Digital Envelope + Digital Certificate are send to receiver.
Digital Certificate

Fig **8.3.2** Working Of PKI

8.4 TRUSTED COMPUTING BASE (TCB)

8.4.1 TCB :

The trusted computing base (TCB) is everything in a computing system that provides a secure environment. This includes the operating system and its provided security mechanisms, hardware, physical locations, network hardware and software, and prescribed procedures.

Typically, there are provisions for controlling access, providing authorization to specific resources, supporting user authentication, guarding against viruses and other forms of system infiltration, and backup of data. It is assumed that the trusted computing base has been or should be tested or verified.

In other words, a given piece of hardware or software is a part of the TCB if and only if it has been designed to be a part of the mechanism that provides its security to the computer system. In operating systems, this typically consists of the kernel (or microkernel) and a select set of system utilities (for example, setuid programs and daemons in UNIX systems).

In programming languages that have security features designed in such as Java and E, the TCB is formed of the language runtime and standard library.

8.4.2 PROPERTIES OF TCB :

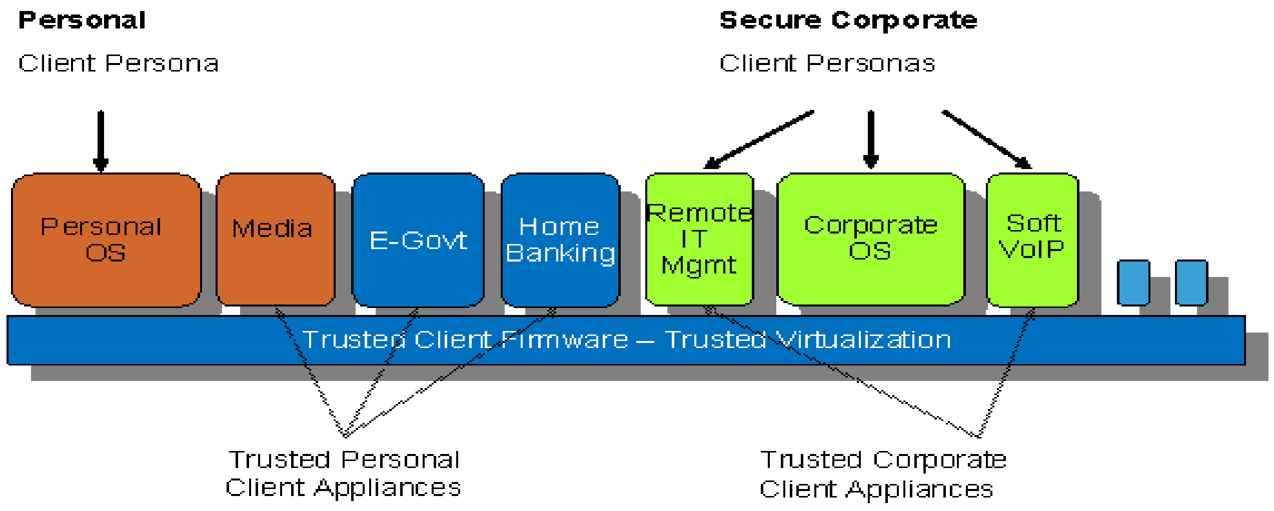


Fig. 8.4.2.1 TCB PROPERTIES

9. TESTING

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet –undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding a yet undiscovered error. A successful test is one that uncovers a yet undiscovered error. Any engineering product can be tested in one of the two ways:

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

9.1 Unit Testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration.

This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

9.2 Integration Testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent.

Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

9.3 Functional Test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	:	identified classes of valid input must be accepted.
Invalid Input	:	identified classes of invalid input must be rejected.
Functions	:	identified functions must be exercised.
Output	:	identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

9.4 System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

9.5 White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

9.6 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

9.7 Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

9.8 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results:

All the test cases mentioned above passed successfully. No defects encountered.

9.9 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

10. CONCLUSION

Enhancing signature trustworthiness is of fundamental importance. We have presented a novel solution to this problem, while dealing with malicious signing application and/or malicious general-purpose OS kernel. To demonstrate its feasibility, we reported a Xen-based implementation and discussed the implementation issues on other Type I hypervisor platforms.

11. FUTURE ENHANCEMENT

Future research directions. There are opportunities for future research.

- First, it is important to reduce, if not eliminate, the reliance on the user to confirm that a message signing operation was indeed initiated by a human user.
- Second, we observe that the user confirmation mechanism offers enhanced source nonrepudiation, which could be exploited to hold insiders (i.e., corrupt authorized users) more accountable. This is because they cannot attribute signatures to the attacks we addressed. As such, it is interesting to characterize to what extent accountability can be gained by deploying ADS.
- Third, it would be very interesting to formalize the type of (modular) security analysis we conducted. It appears that both the cryptographic framework and the Dolev-Yao framework are not applicable to this type of analysis. One challenge is that the security properties very much depend on the implementation details.
- Fourth, it would be interesting to extend ADS to accommodate more cryptographic applications.

12. BIBLIOGRAPHY

[1] S. Goldwasser, S. Micali, and R. Rivest, "A Digital Signature Scheme Secure against Adaptive Chosen-Message Attacks," SIAM J. Computing, vol. 17, pp. 281-308, Apr. 1988.

[2] A. Akavia, S. Goldwasser, and V. Vaikuntanathan, "Simultaneous Hardcore Bits and Cryptography against Memory Attacks," Proc. Sixth Theory of Cryptography Conf. Theory of Cryptography (TCC), O. Reingold, ed., pp. 474-495, 2009.

[3] Y. Desmedt and Y. Frankel, “Threshold Cryptosystems,” Proc. Ninth Ann. Int’l Cryptology Conf. Advances in Cryptology, pp. 307-315, 1990.
DAI ET AL.: IMPROVING DATA RELIABILITY THROUGH ASSURED DIGITAL SIGNING 849

[4] R. Ostrovsky and M. Yung, “How to Withstand Mobile Virus Attacks (Extended Abstract),” Proc. 10th Ann. ACM Symp. Principles of Distributed Computing (PODC ’91), pp. 51-59, 1991.

[5] R. Anderson, “On the Forward Security of Digital Signatures,” technical report, 1997.
[6] M. Bellare and S. Miner, “A Forward-Secure Digital Signature Scheme,” Crypto ’99: Proc. 19th Ann. Int’l Cryptology Conf. Advances in Cryptology, M. Wiener, ed., pp. 431-448, 1999.

[7] Y. Dodis, J. Katz, S. Xu, and M. Yung, “Strong Key-Insulated Signature Schemes,” Proc. Sixth Int’l Workshop Theory and Practice in Public Key Cryptography (PKC ’03), pp. 130-144, 2003.

[8] G. Itkis and L. Reyzin, “SiBIR: Signer-Base Intrusion-Resilient Signatures,” CRYPTO ’02: Proc. 22nd Ann. Int’l Cryptology Conf. Advances in Cryptology, pp. 499-514, 2002.

[9] B. Yee, “Using Secure Coprocessors,” PhD thesis, Carnegie Mellon Univ., May 1994.
[10] P. Loscocco, S. Smalley, P. Muckelbauer, R. Taylor, S. Turner, and J. Farrell, “The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments,” Proc. 21st Nat’l Information Systems Security Conf. (NISSC ’98), 1998.

[11] S. Xu and M. Yung, “Expecting the Unexpected: Towards Robust Credential Infrastructure,” Proc. Int’l Conf. Financial Cryptography and Data Security (FC ’09), Feb. 2009.

[12] Z. Wang and X. Jiang, “Hypersafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity,” Proc. IEEE Symp. Security and Privacy, pp. 380-395, 2010.

[13] A. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. Skalsky, “Hypersentry: Enabling Stealthy In-Context Measurement of Hypervisor Integrity,” Proc. ACM Conf. Computer and Comm. Security, pp. 38-49, 2010.

[14] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D.C.P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, “Sel4: Formal Verification of an os Kernel,” Proc. ACM SIGOPS 22nd Symp. Operating Systems Principles (SOSP ’09), 2009.

[15] U. Steinberg and B. Kauer, “Nova: A Microhypervisor-Based Secure Virtualization Architecture,” Proc. Fifth European Conf. Computer systems (EuroSys ’10), 2010.

[16] J. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "TrustVisor: Efficient TCB Reduction and Attestation," Proc. IEEE Symp. Security and Privacy, 2010.

[17] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "Kvm: The Linux Virtual Machine Monitor," Proc. Linux Symp., 2007.

[18] C. Systems, "Xen Project." http://www.xen.org/, 2011.

[19] C. Systems, "Citrix Xenserver." http://www.citrix.com/ xenserver, 2011.

[20] R. Oglesby and S. Herold, VMware ESX Server: Advanced Technical Design Guide, Advanced Technical Design Guide Series. The Brian Madden Company, 2005.

[21] C. Chauba, "The Architecture of Vmware Esxi," VMware White Paper, 2008.

[22] Microsoft, "Windows Server 2008 r2 Hyper-V." http://www.microsoft.com/en-us/server-cloud/windows-server/hyper-v.aspx, 2010.

[23] T.C. Group https://www.trustedcomputinggroup.org/, 2012.

[24] E. Brickell, J. Camenisch, and L. Chen, "Direct Anonymous Attestation," Proc. 11th ACM Conf. Computer and Comm. Security (CCS '04), pp. 132-145, 2004.

[25] R. Ta-Min, L. Litty, and D. Lie, "Splitting Interfaces: Making Trust between Applications and Operating Systems Configurable," Proc. Seventh Symp. Operating Systems Design and Implementation (OSDI '06), pp. 279-292, 2006.

[26] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A Virtual Machine-Based Platform for Trusted Computing," ACM SIGOPS Operating Systems Rev., vol. 37, no. 5, pp. 193-206, Oct., 2003.

[27] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," technical report, EECS Dept., Univ. of California, Berkeley, Feb. 2009.

[28] H. Lagar-Cavilla, J. Whitney, A. Scannell, P. Patchin, S. Rumble, E.de Lara, M. Brudno, and M. Satyanarayanan, "Snowflock: Rapid Virtual Machine Cloning for Cloud Computing," Proc. Fourth ACM European Conf. Computer Systems (EuroSys '09), pp. 1-12, 2009.

Reference For Journal:
IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, VOL. 9, NO. 6, NOVEMBER / DECEMBER 2012