5-2012

Security and Performance Verification of Distributed Authentication and Authorization Tools

Seok bae Yun
Clemson University, seokbae.yun@gmail.com

Security and Performance Verification of Distributed Authentication and Authorization Tools

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Seok Bae Yun
May 2012

Accepted by:
Dr. Richard R. Brooks, Committee Chair
Dr. Haying Shen
Dr. Jill Gemmill

Abstract

Parallel distributed systems are widely used for dealing with massive data sets and high performance computing. Securing parallel distributed systems is problematic. Centralized security tools are likely to cause bottlenecks and introduce a single point of failure. In this paper, we introduce existing distributed authentication and authorization tools. We evaluate the quality of the security tools by verifying their security and performance.

For security tool verification, we use process calculus and mathematical modeling languages. Casper, Communicating Sequential Process (CSP) and Failure Divergence Refinement (FDR) to test for security vulnerabilities, Petri nets and Karp Miller trees are used to find performance issues of distributed authentication and authorization methods.

Kerberos, PERMIS, and Shibboleth are evaluated. Kerberos is a ticket based distributed authentication service, PERMIS is a role and attribute based distributed authorization service, and Shibboleth is an integration solution for federated single sign-on authentication. We find no critical security and performance issues.

Dedication

I would like to dedicate this master thesis to my father Hyo Hyun Yun, my mother Byung Hee Han, and my brother Jeoung Bae Yun. There is no doubt in my mind that without continued support and counsel from my family I could not have completed this process.

Acknowledgments

I would like to acknowledge the instruction and guidance of Dr. Richard R. Brooks. Every recommendation of his was for the best interest of his graduate students. He has given me a deep appreciation and detail of this subject.

I would also like to acknowledge the support and assistantship given me by Dr. Haying Shen and Dr. Jill Gemmill. Especially, I appreciate that Dr. Gemmill had a review on my modeling of security protocol models.

I would also like to thank my family for their encouragement and love throughout my graduate study, and their belief in me. Also, heartfelt thanks to all my friends, lab colleagues and the faculty of the ECE department. Finally, all works on this thesis is supported by the National Science Foundation, under grant NSF-OCI 1064230 EAGER: Collaborative Research: A Peer-to-Peer based Storage System for High-End Computing, and ORNL 4000111689 Novel Software Storage Architectures.

Table of Contents

Title Page . . . . . **i**

Abstract . . . . . **ii**

Dedication . . . . . **iii**

Acknowledgments . . . . . **iv**

List of Tables . . . . . **vii**

List of Figures . . . . . **viii**

1 Introduction **1**
1.1 Assumptions 3

2 Related Work **5**
2.1 Distributed File Systems 5
2.2 Related security art 8
2.3 Distributed systems authentication 10

3 Security Verification **18**
3.1 Casper 20
3.2 CSP 24
3.3 FDR 26

4 Performance Evaluation **29**
4.1 Petri nets 30
4.2 Similarity between CSP and Petri net 32
4.3 Reachability and Karp Miller Trees 32
4.4 Connectivity Matrix 35
4.5 Deadlocks and Livelocks 36
4.6 Bottleneck Analysis 37

5 Distributed Authentication and Authorization **38**
5.1 Kerberos 39
5.2 PERMIS 41
5.3 Shibboleth 43

6 Implementation 46
6.1 UML sequence diagram 46
6.2 Security verification 48
6.3 Performance Evaluation 52

7 Test Results 57
7.1 Security test result 57

8 Conclusions and Future Work 60

Appendices 61
A Security Test Results on FDR 62
B Petri Net Graphs 65
C Karp-Miller Trees Graphs 68
D Casper Scripts 71

Bibliography 85

List of Tables

2.1 Comparision of Distributed Authentication Tools 16
2.2 Comparision of Distributed Authorization Tools 17

6.1 Mathematical Notation for Security Protocols 47
6.2 Mathematical Notation used for Kerberos 49
6.3 The Structure of SAML Attribute Assertion in PERMIS [22] 51
6.4 Attribute Query Message (AQM) Common Syntax [24] 52
6.5 Attribute Response Message (ARM) Common Syntax [24] 53

7.1 Security Test result of Kerberos 58
7.2 Security Test result of PERMIS 58
7.3 Security Test result of Shibboleth 58
7.4 Livelock and Deadlock Test Results 59

List of Figures

2.1 The Authorization Process of Light Weight File System 6
2.2 The Authorization of Parallel Virtual File System 7
2.3 The Authorization of Lustre 8
2.4 The Authorization of Pansas File System 9

3.1 Security verification procedure of Needham-Schroeder protocol using Casper, CSP, and FDR 19
3.2 Compilation result of Needham-Schroeder protocol in Casper compiler . . . 24
3.3 Needham-Schroeder protocol FDR test result 26

4.1 Basic components and an example of Petri net model 30
4.2 Basic Event Structures Modeling in CSP and Petri Net [30] 32
4.3 Transition firing examples of Petri net model 33
4.4 The reachability tree of the Petri net example 35

5.1 The system diagram of Kerberos 40
5.2 The system diagram of PERMIS 42
5.3 The system diagram of Shibboleth 44

6.1 Overall process of security verification and performance evaluation 47
6.2 System level sequence diagram of Kerberos 50
6.3 System level sequence diagram of PERMIS 51
6.4 System level sequence diagram of Shibboleth 54

1 Karp-Miller Tree of Kerberos with Two initial tokens 68
2 Karp-Miller Tree of PERMIS with Two initial tokens 69
3 Karp-Miller Tree of Shibboleth with Two initial tokens 70

Chapter 1

Introduction

Parallel distributed computing, a.k.a high performance computing, has been used to solve many important compute-intensive problems such as quantum physics, weather forecasting, climate research, molecular modeling, or nuclear weapon simulations. Hardware components of supercomputers are not different from those of personal computers, since modern parallel distributed systems are made up of multiple servers using commodity hardware for cost efficiency. Performance is achieved by interactions between hundreds to millions of computing nodes.

Although the computing power of parallel distributed system is strong enough, slower storage nodes or bottlenecks on the communication between them can harm overall system performance. Therefore, larger and faster file systems are required for modern high performance computing. Distributed file systems provide both high performance and robust file access to parallel distributed system. Distributed file systems need to pay more attention to possible security threats than do centralized file systems. Applying distributed authentication and authorization could be a good solution for distributed file systems to maintain security while delivering exabyte performance, but we must check whether existing authentication and authorization tools have any negative impact on performance or security.

Distributed file system must assure security to prevent altering, forging or sniffing of

data. Security requirements for parallel distributed systems are quite different from those of personal computers even though the hardware components of supercomputers are made of commodity hardware, and Linux is commonly used as an O/S, since centralized security tools are likely to cause bottlenecks and introduce a single point failure.

Standardized cryptography protocols are typically more secure than newly proposed concepts, mainly because they have been thoroughly peer reviewed. Most distributed systems on the internet have similar needs and a set of standardized tools has been developed to fulfill the performance requirement of distributed file system. On the other hand, distributed approaches which rely on a single root of trust are almost certain to become a performance bottleneck. We must check whether the authentication and authorization tools have performance or security problems.

To secure the system, the security tools maintain security attributes. These security attributes guarantee that:

- participants are authenticated; access rights are verified at session initiation and revoked as necessary;
- transactions cannot be repudiated;
- and a reliable audit trail of security events is maintained.

The network needs to provide a sensible integration of cryptographic primitives into file system communications; guarding against man-in-the-middle attacks, replay attacks, eavesdropping, and data tampering. Cryptographic primitives are needed for data at rest security, especially when files move to tertiary storage. Authentication and authorization tools typically rely on secure cryptographic key distribution protocols.

To verify security properties of a given security protocol, we use Communicating Sequential Processes (CSP) formalisms [33, 51] as implemented in the Failure Divergence Refinement (FDR) model checker. CSP is a mathematical framework for description and analysis of systems with component interactions via message exchange. FDR is an automated model checker which can determine whether the security property is fulfilled or not.

If not, a counter example is returned by FDR. Constructing the CSP model of a security protocol is somewhat tricky. A compiler called Casper was developed [32] to ease model development. With Casper, only high level information of a security protocol to be checked is needed. Casper automatically constructs the CSP model for FDR. We use a three message version of Needham-Schoroeder message exchange protocol as an example for proving the validity of our security and performance verification methods.

The Petri net is a mathematical model for describing discrete distributed systems and it has been widely used for performance analysis on different fields. Any event structure that can be modeled by CSP can also be modeled by Petri nets. By evaluating the Petri net model of security protocols, we measure the performance of security protocols. Karp Miller tree of a stochastic Petri net (SPN) is identical to the underlying Petri net model. By using the reachability tree, we can directly find deadlocks and livelocks.

We provide details on CSP, FDR, and Casper script used for our security verification methods in chapter 3. In chapter 4 we discuss on the theories of Perti net and Karp-Miller Tree used for performance verification. We briefly introduce Kerberos, PERMIS and Shibboleth in Chapter 5. Chapter 6 shows how we implement the test software based on the theories introduced in 3 and 4. Then, we show our results and conclusion in chapter 7 and 8.

1.1 Assumptions

We assume the storage system executes in an environment where low level communications security is assured. Address spoofing and message security are not relevant to this work. The system either uses an architecture where these attacks are not feasible, or security is provided by a lower level of the implementation stack. For example, a wide area network implementation could use IPV6 with IPSec. In which case, the networking layer secures the communications. We do not explicitly consider side-channel attacks, since these vulnerabilities are orthogonal to this work. In environments where side-channel attacks are

relevant, countermeasures, such as equalization of resource consumption, are usually best handled using some combination of compiler technologies and the physical or MAC layers of the architecture [10]. The specific assumptions we make are:

- File System
  - System implementation does not introduce additional vulnerabilities.
  - Storage nodes are not corrupted
  - Eavesdropping on packets is not viable.
  - Client nodes may be corrupted.
  - Infinite number of servent nodes possible.
- Network
  - No sniffing/spoofing.
  - Channel security assured at another level of abstraction (e.g. IPSec or SSL/TLS).
- Security
  - Client login on local node outside of scope.
  - Side-channel attacks outside of scope.
  - Data scavenging outside of scope.
  - Physical attacks out of scope.

Chapter 2

Related Work

Exabyte-scale distributed file systems need to reliably store and retrieve information at rates exceeding 1018 bytes per second. Reliable storage and retrieval implies system enforcement of basic security guarantees. Since enforcing security guarantees is common to all information processing systems, it is only reasonable to leverage existing security enforcement tools. In this chapter, We skim over a brief overview on existing distributed file systems, and related security art. It then proposes a general authentication and authorization architecture for the proposed system.

2.1 Distributed File Systems

2.1.1 LWFS

The Light Weight File System (LWFS) controls access to file containers through the use of capabilities that provide proof of authorization for file access. Capabilities are provided by processes possessing the proper credentials. Credentials serve as proof of user authorization by a trusted external mechanism. Credentials and capabilities may be transferred between processes as needed [45]. The LWFS architecture has authentication and authorization servers that are separate from file storage servers [46]. This separation adds extra components to the system architecture and may eventually be a performance bottle-

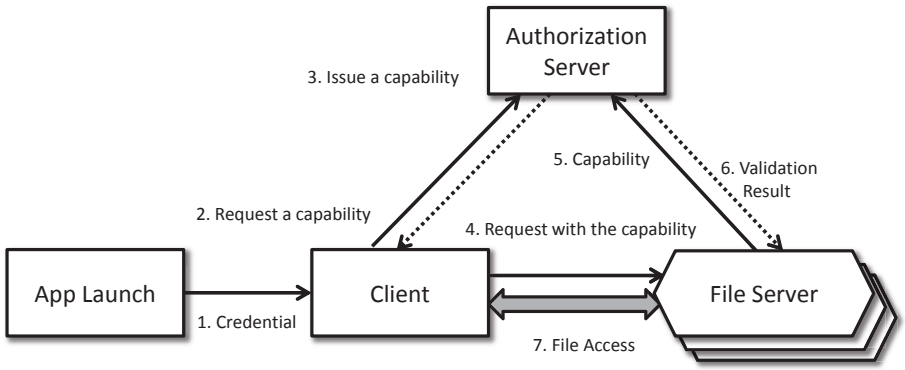


Figure 2.1: The Authorization Process of Light Weight File System

neck.

2.1.2 PVFS

The Parallel Virtual File System (PVFS) is a high performance distributed file system for Linux clusters. PVFS security is enforced through digital signing. PVFS metadata contains the information needed to verify access rights. Metadata is retrieved once by a client and then reused as needed. Authentication is performed first by the local operating system and then public key cryptography is used for authentication within PVFS. For the sake of performance, X.509 and Assertion Markup Language (SAML) data structures are not used. For non-repudiation, data signing is done using RSA [2]. For the sake of performance, the RSA key lengths used are relatively short.

2.1.3 GPFS security

IBMs General Parallel File System (GPFS) is a high performance distributed file system [50]. Security is maintained by using SSL and RSA public key authentication [3]. RSA public keys are provided to system administrators for mounting volumes. SSL typically uses public key primitives to generate symmetric keys that are used to maintain security

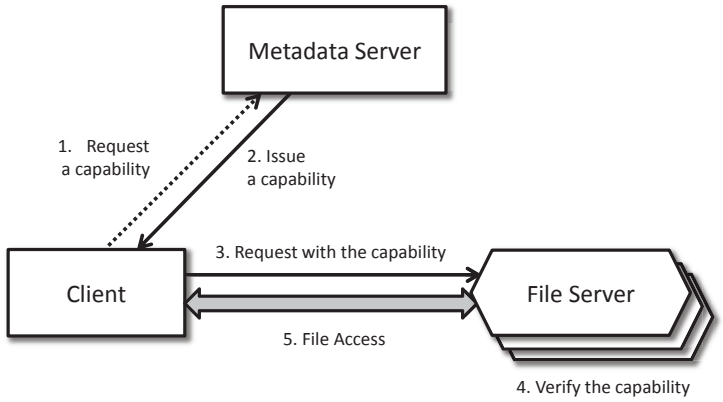


Figure 2.2: The Authorization of Parallel Virtual File System

for the life of the session.

2.1.4 Lustre

Lustre is a massively parallel file system for Linux Clusters. Lustre [Su-Qin 2010] supports two authentication mechanisms: Kerberos and Public Key Infrastructure (PKI) [Braam 2004]. Kerberos uses a centralized authentication server that must be on-line continuously. In its PKI implementation, X.509 certificates are generated by a trusted authority. To access the file system, both the client and Metadata Server (MDS) exchange messages and access an LDAP directory. The MDS then transmits information that the client can use to access an Object Storage Target (OST).

2.1.5 Panasas file system

The Panasas file system uses a symmetric key approach within the Generic Security Services Application Programming Interface (GSS-API). Multiple implementations could be consistent with this Object-based Storage Device (OSD) standard. In [Ko 2006] an authentication protocol is detailed with communications between six entities using a set of seven hierarchical symmetric keys. [Leung 2006] has a different approach where multiple

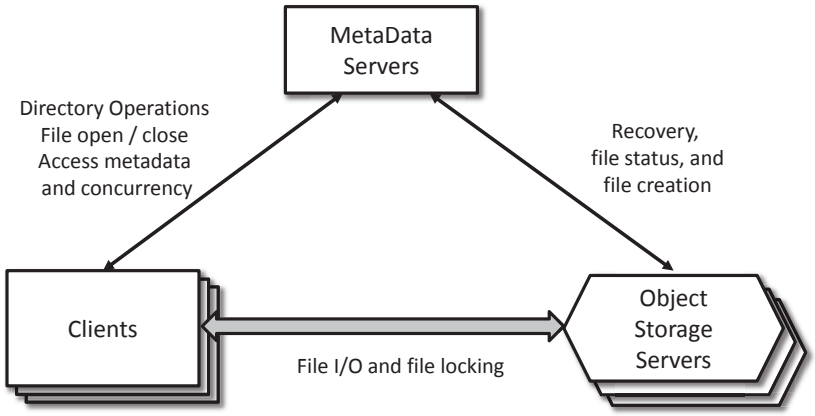


Figure 2.3: The Authorization of Lustre

MDSs work in parallel to respond to client authentication requests. In [Leung 2006] MDSs return access handles that are capabilities allowing clients to access OSD. A major vulnerability to this approach is that both client and MDS start with an a priori known symmetric key [Oldfield 2006a]. The compromise of any client would destroy the security model. In addition, access revocation is problematic.

2.2 Related security art

Distributed system security is important for many applications. Multiple tools exist. Most implement established standards. Standards are adapted after rigorous peer review. It is preferable to rely on peer-reviewed security standards rather than new security concepts whenever possible. New security concepts are frequently derided as security through obscurity [Lam 2004]. They are often brittle and subject to flaws that the original designer did not foresee. This section reviews the tools most relevant to the P2P file system.

Many standards exist for handling security credentials. These standards allow security systems to interact transparently. They reduce the need to implement parsers, error checking, and other common support functions. Unfortunately, data structure standards

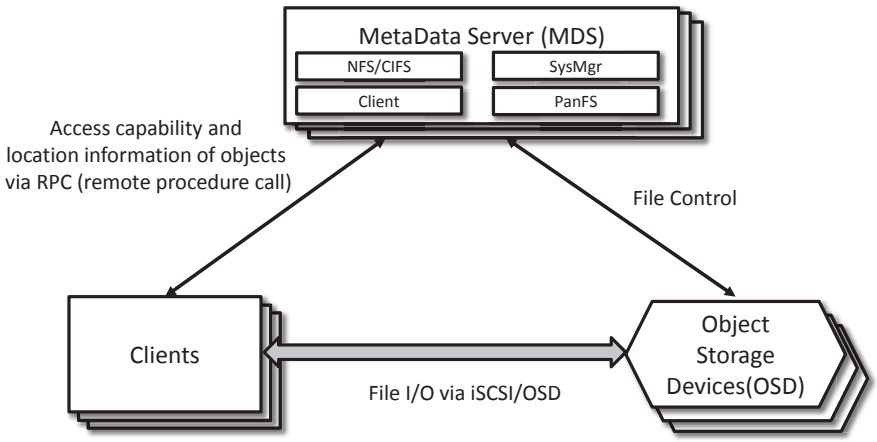


Figure 2.4: The Authorization of Pansas File System

are typically designed by committees seeking to foresee all possible eventualities. They are often bloated, which may have performance implications for an Exabyte file system.

2.2.1 XML based data structures

Numerous security data structure standards have been issued by OASIS based on XML. SAML is a data structure for exchanging authentication and authorization information between security domains [Cantor 2005]. XACML (eXtensible Access Control Markup Language) is a language for expressing access control policies. XRML (eXtensible rights Markup Language) is a language for expressing access rights for digital content [XrML 2010]. XRML and XACML are very similar.

- Pros
  - Strong authentication based on PKI
  - Widely used as a standard
- Cons
  - Single point of failure vulnerability: It requires continuous availability of a central

server.

  - Heavy traffic bottleneck
  - Heavy processing
  - Attacker still can construct bad certificates.

2.3 Distributed systems authentication

This brief discussion of distributed authentication standards will mention the major tools; presented roughly in chronological order. We concentrate on the aspects of each tool that are most relevant to this project.

2.3.1 Kerberos

Kerberos uses Needham-Schroeder key distribution [Needham 1978] to authenticate users. To access a service, the client requests a ticket [Neuman 1994]. Since a centralized authentication server must be continuously available, Kerberos would introduce a single point of failure to this system and would quickly become a performance bottleneck. Further discussions on Kerberos will continue in section 6.

- Pros
  - Less computing power if it only uses symmetric keys.
  - Flexible on cryptography
- Cons
  - Designed for trustworthy server-client models, not for P2P system.
  - Heavy traffic bottleneck.
  - Vulnerability on symmetric key encryption: if one key has been lost, whole system will be vulnerable in certain amount of time.

- Single point of failure vulnerability: It requires continuous availability of a central server.
- Kerberos has strict time requirements, which means the clocks of the involved hosts must be synchronized within configured limits.

2.3.2 PGP Web of trust

The PGP web of trust tries to remove the need for a single root of trust and centralized authentication servers. In this approach, public key certificates are signed by many authorities that attest to their validity [Blaze 1996]. As long as each participant is diligent in verifying user identity, this process should be robust. Problems exist with collusion attacks [Lenstra 2005], certificate revocation, and users with disjoint sets of colleagues.

- Pros
  - Strong authentication based on PKI
  - Good for decentralized network
  - Less or no traffic bottleneck
  - Easy to manage
- Cons
  - Attackers also can make fake public keys and issue them.
  - PGP methods still require a lot of processing power.

2.3.3 Group keys with key management trees

An alternative to both the use of public keys and the need for a continuously available centralized server is the use of binary key trees [Poovendran 1999, Pillai 2006, Brooks 2007, Brooks 2007a, Brooks 2009] for key management. The entire workgroup shares a common symmetric key for securing communications. A binary tree structure of Key-Encryption-Keys (KEK) is created where each client is a leaf node and there is a symmetric

KEK associated with each node of the tree. Each of the n clients therefore has to store keys. The key management server refreshes the communications keys periodically using messages secured using the KEK. It is possible to efficiently revoke system access for any subset of clients by modifying the group key and excluding that subset of clients. In [Pillai 2006], we show how this can be securely implemented for a set of peer nodes to combat cloning, Sybil, and Byzantine Generals attacks. If key management nodes have overlapping sets of clients, it is possible to detect malfeasance by an isolated key management node. The use of secure key server selection schemes [Pirretti 2005, Pirretti 2006] can also make this approach immune to collusion attacks.

- Pros
  - Fast key encryption and decryption
  - Distributed authentication.
  - Less bottlenecks on network.
  - Good for distributed system.
- Cons
  - Not good for unstructured P2P.
  - Vulnerability on symmetric key encryption: if one key has been lost, whole system will be vulnerable in certain amount of time.

2.3.4 LDAP

X.509 was designed to define secure identifier entries in a universal X.500 (DAP) directory system for the X.400 messaging system. The X.500 and X.400 standards were never fully implemented. LDAP (Lightweight DAP) is a distributed directory system that implements major portions of the X.500 standard [Howes 1995]. It forms a bridge between the ITU and IETF standards families. LDAP can be used as a tool for organizing and retrieving authorization information [Wahl 2000]. LDAP communications use TCP, which

may be secured using either TLS or Kerberos. The use of TCP may be problematic in this application. The rest of the file system is expected to be based on UDP to allow more flexibility. LDAP is also subject to injection attacks that may result in data disclosure, modification of the LDAP directory, and corruption of LDAP data [Alonso 2008]. The LDAP design has also been criticized for not maintaining referential integrity [Blaha 2005].

- Pros
  - LDAP directory can be accessed from any computing platform.
  - LDAP servers are simple to install, easily maintained, and easily optimized.
  - LDAP servers can replicate either some or all of their data via push or pull methods, allowing you to push data to remote offices, or increase security.
  - LDAP allows users delegate read and modification authority based on ACL.
- Cons
  - It is not well suited for storing data where changes are frequent.

2.3.5 DIAMETER

DIAMETER is an authentication and authorization protocol that is the successor to RADIUS (Remote Authentication Dial In User Service). It relies on TCP sessions secured using either TLS or an existing IPsec tunnel [Calhoun 2003]. TLS vulnerabilities based on X.509 infrastructure are discussed in [Brooks 2010]. Similarly, IPsec may be vulnerable to X.509 issues and is very vulnerable to Denial of Service attacks [Nikov 2006]. Each hop of the base DIAMETER protocol is secure, but unless the end-to-end variant is used proxies can execute man-in-the-middle attacks. The DIAMETER node serves as a centralized repository of authentication information.

2.3.6 TACACS+

TACACS+ (Terminal Access Controller Access-Control System Plus) is a Cisco proprietary access and authentication system [Cisco 2010]. TACACS+ and DIAMETER are considered current authentication and authorization approaches. TACACS+ has a number of security issues [Peslyak 2010]. It has integrity checking issues, is vulnerable to replay attacks, its encryption can be compromised if session IDs are not unique, session collisions can reveal passwords, and a lack of padding compromises password strength [Young 2004].

2.3.7 Shibboleth

Shibboleth is an Internet2 Middleware Initiative project that is for federated identity-based authentication and authorization infrastructure based on SAML. Federated identity allows for information about users in one security domain to be provided to other organizations in a federation [24]. This allows for cross-domain single sign-on and removes the need for content providers to maintain user names and passwords [42].

- Pros
  - Organizational Single Sign-on System
  - Controlled Information Release
  - Federated Access
  - Virtual Identity Provider
- Cons
  - Heavy traffic bottleneck.
  - Single point of failure vulnerability: It requires continuous availability of a central server.

2.3.8 PERMIS

PERMIS supports the distributed assignment of both roles and attributes to users by multiple distributed attribute authorities, unlike centralized assignment of roles to users. PERMIS provides a cryptographically secure Privilege Management Infrastructure (PMI) using public key encryption technologies and X.509 attribute certificates to maintain user attributes [Chadwick 2003]. PERMIS does not provide any authentication mechanism, but leaves it up to the application to determine what to use. (In earlier version, PERMIS only supported X.509) Currently, PERMIS uses LDAP as a network accessible repository for storing policies and credentials [Permis 2010]. PERMIS does not do authentication, but may be integrated with authentication systems, such as DIAMETER or Shibboleth. We further discuss Permis as our proposed authorization tool in section 5.

- Pros
  - PERMIS can be integrated into virtually any application and any authentication scheme like Shibboleth (Internet2), Kerberos, username/passwords, and PKI.
  - Less computing power than PGP.
  - Less bottlenecks on network.
  - Good for P2P system by distributed authorization servers.
  - It directly support XACML and SAML
- Cons
  - No authentication mechanism exists. It should be used with other methods, but it may require more computing power than other single security mechanism.
  - Vulnerability on symmetric key encryption: if one key has been lost, whole system will be vulnerable in certain amount of time.
  - LDAP vulnerability

	PKI	**Kerberos**	**Athens**
Authentication Mechanism	Digital certificates / signatures	Tickets	Usernames
Single Sign On	Proxy certificates	Can be provided	Partial through usernames
Authentication Delegation	Through proxy certificates	Cross-realm trust configurations	Not provided
Authentication Usability	Cumbersome process of acquiring, using and managing the certificate	User friendly as the process of ticket generation is hidden from user	User-friendly process of getting Athens usernames

Table 2.1: Comparision of Distributed Authentication Tools

W. Jie et. al. compared the distributed authentication and authorization tools in [29]. We examine the paper to compare and decide which security tools are appropriate to our design purpose. The table 2.1 and 2.2 are from the paper.

	Grid-map file	**CAS**	**VOMS**	**PERMIS**	**Akenti**
Authorization Model	User identity	Role	Attribute	Attribute	Attribute
Authorization Mode	Distributed	Centralized (CAS server)	Centralized (VOMS server)	Distributed	Distributed
Authorization Delegation	No	Partially	Partially	Yes	Yes
Authorization Granularity	Coarse-grained	Fine-grained when each SP defines fine-grained policy	Fine-grained when each SP defines fine-grained policy	Multi-grained	Multi-grained
Performance Scalability	Not good (scalability issue)	Not good (SPF & Scalability issue)	Better than CAS	Good	Good
Authorization Manageability	Manage whole grid-map & heavy load	Need fine-grained policies	Need fine-grained policies	Independent role-based management	Achieved using delegation capabilities
Authorization Usability	Easy-to-use but lack of policy expression	Plain extension to proxy certificates	Not addressed	Easy-to-use in XML	GUI interface
Credential Confidentiality	Not applicable	Kept & managed by Authorization Infrastructure	Kept & managed by Authorization Infrastructure	Kept & managed by Authorization Infrastructure	Kept & managed by Authorization Infrastructure
Communication Confidentiality	TLS/MLS	SAML	SAML	SAML/XACML	SAML

Table 2.2: Comparision of Distributed Authorization Tools

Chapter 3

Security Verification

Security protocols are designed to assure a set of security properties; confidentiality, authentication, integrity, non-repudiation, anonymity, access control, availability [10]. The protocols involve cryptographic operations such as hash functions, encryption, and digital signatures. Some protocols such as TLS, Kerberos and AAA may require the participation of a third trusted party. They always assume the underlying cryptographic mechanisms are perfect, but we need to ensure the protocols are valid.

Security protocol validation has been an active research topic. Much of this work uses various types of logic, such as BAN-logic [58, 35, 57, 25], linear temporal logic [19], and other variants [9, 16, 17, 18]. Using logic, it is possible to check whether or not a security protocol provide required security. If it fails to provide a security property, usually a counter example will be generated.

Another way to analyze security prosperities utilizes CSP [33, 51, 55, 53, 48] and the FDR model checker. CSP, proposed by C. A. R. Hoare in 1985 [28], is a mathematical framework for describing and analyzing systems consisting of components interacting via the exchange of messages [28, 47, 52]. FDR is an automated model checker [1].

For analyzing a security protocol against a given security property, a CSP model of the security protocol is created, which include message flows of the security protocol, the security property to be checked, and intruders with explicit capabilities, such as eavesdrop-

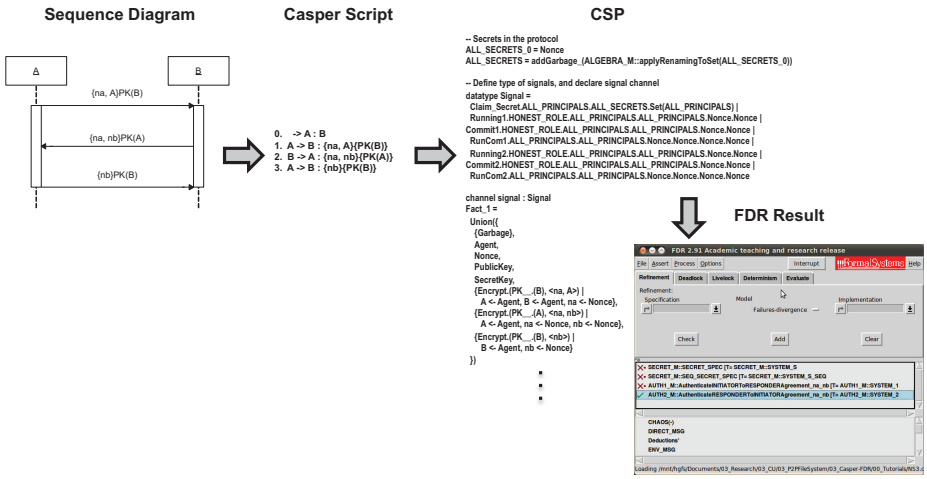


Figure 3.1: Security verification procedure of Needham-Schroeder protocol using Casper, CSP, and FDR

ping, forging, password cracking, and so on. Once the protocol is described by CSP, FDR can analyze the effect of possible threats on the protocol. The result of FDR is that the security property is either fulfilled or failed. If failed, a counter example is returned.

Casper is a script language developed for constructing CSP code for a security protocol[49]. With Casper, only a high level description of a security protocol is needed. Casper automatically constructs FDR code. Figure 3.1 shows how security analysis can be done with Casper, CSP, and FDR. In this section, we explain our security analysis procedures using a simple example of Needham-Schroeder message exchange protocol, introduced in [41].

The security verification procedure is as follows. First we gather cryptography and message passing information from design documents, source codes, and papers, From this information, we can draw a sequence diagram of the security protocol. Then, Casper scripts of the security protocol can be written based on the sequence diagram. Next, we compile Casper scripts to translate them into CSP. Security protocols written in CSP can be analyzed by FDR. The sequence diagram will also be used for performance analysis.

3.1 Casper

The Casper scripting language and its compiler were first introduced by G. Lowe in [32]. The scripting language is designed for describing security protocols in a simple, intuitive, and abstract notation. The Casper compiler translates Casper Script into CSP.

3.1.1 Casper script

Casper script consists of eight different sections to precisely describe protocols. Each section starts with '#' notation. We show an example Needham-Schroeder protocol from [34]. Full Casper script of the protocol shown in figure 3.1 is listed as follows.

```
-- Needham Schroeder Public Key Protocol, 3 message version

#Protocol description
0.    -> A : B
1.  A -> B : {na, A}{PK(B)}
2.  B -> A : {na, nb}{PK(A)}
3.  A -> B : {nb}{PK(B)}

#Free variables
A, B : Agent
na, nb : Nonce
PK : Agent -> PublicKey
SK : Agent -> SecretKey
InverseKeys = (PK, SK)

#Processes
INITIATOR(A,na) knows PK, SK(A)
RESPONDER(B,nb) knows PK, SK(B)

#Specification
Secret(A, na, [B])
Secret(B, nb, [A])
Agreement(A,B,[na,nb])
```

```
Agreement(B,A,[na,nb])

#Actual variables
Alice, Bob, Mallory : Agent
Na, Nb, Nm : Nonce

#Functions
symbolic PK, SK

#System
INITIATOR(Alice, Na)
RESPONDER(Bob, Nb)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {Alice, Bob, Mallory, Nm, PK, SK(Mallory)}
```

We introduce the role of each section and the Casper syntax used in this example. Please refer to [32, 34] for detailed Casper syntax. The line which starts with '–' is a comment line and ignored by Casper. The script has two parts; protocol description and system definition. The protocol description defines the security and communication protocol. It has four individual sections; protocol description, free variables, processes, and specification.

```
#Protocol description
0.    -> A : B
1.  A -> B : {na, A}{PK(B)}
2.  B -> A : {na, nb}{PK(A)}
3.  A -> B : {nb}{PK(B)}
```

"#Protocol description" models the message flow in Figure 3.1. In this model, we assume that A is a client, and B is a server. -> represents message flow direction. "0. -> A : B" is included to start the protocol, representing that the client A obtains the identity of server B from the environment. Message "1. A -> B : {na, A}{PK(B)}" means client A sends server B a nonce *na* and B's identity encrypted by server B's public key. Since the

three message version of Needham-Schroeder protocol uses public key encryption and each agent has its own secret key, the message sent by A can be decrypted by B using B's secret key. The nonce is a random number that is used only once [41]. Message 2 and 3 use the same notation with message 1. "2. B -> A : {na, nb}{PK(A)}" means B sends *na* and newly generated *nb* encrypted by A's public key. Message "3. A -> B : {nb}{PK(B)}" is for B to confirm the nonce *nb* generated by B itself.

```
#Free variables
A, B : Agent
na, nb : Nonce
PK : Agent -> PublicKey
SK : Agent -> SecretKey
InverseKeys = (PK, SK)
```

"#Free variables" is a variable list which includes agents as servers or clients, nonces for cryptographic messages, and cryptographic keys for encryption and decryption. "Inverse Key = (PK, SK)" means the system uses asymmetric key cryptography.

```
#Processes
INITIATOR(A,na) knows PK, SK(A)
RESPONDER(B,nb) knows PK, SK(B)
```

"#Processes" shows various information on the agents of the protocol using plain English statements. The statements in the example show who knows which cryptographic keys.

```
#Specification
Secret(A, na, [B])
Secret(B, nb, [A])
Agreement(A,B,[na,nb])
Agreement(B,A,[na,nb])
```

The security properties to be checked are specified in "#Specification". We check the secrecy of *na* and *nb*. "Secret(A, na, [B])" means that "A thinks that *na* is a secret that can be known to only himself A and B". Agreement lines are for agreement authentication

specifications; that A . "Agreement(A,B,[na,nb])" means "If A completes a run of the protocol, apparently with B, then B has been running the protocol, apparently with A: further, the two agents agree upon the roles each took and upon the values of the nonces *na* and *nb*; and there is a one to one relationship between such runs of A and those of B." [49]

The system definition part describes which agents should be tested against which malicious attacks. The part includes four sections; actual variables, functions, the system, and the intruder.

The "#Actual variables" section defines which items should be dealt with on the security verification. In this system, the FDR software verification tool checks three agents and three nonces.

```
#Actual variables
Alice, Bob, Mallory : Agent
Na, Nb, Nm : Nonce
```

The "#Function" section describes any functions used by the agents in the protocol description. "symbolic" means that Casper generates its own values to show the result of the function applications.

```
#Functions
symbolic PK, SK
```

The "#System" section defines who will have the role of initiator, responder, or server in the protocol. In addition, it defines who uses what in the protocol. For example, "INITIATOR(Alice, Na)" means that initiator *Alice* will use *Na* in the protocol.

```
#System
INITIATOR(Alice, Na)
RESPONDER(Bob, Nb)
```

In the script, the "#Intruder Information" section models the capacity of an attacker. It suggests that the attacker compromises a legitimate user and obtained that user's credentials, including the secret key.

```
#Intruder Information
```

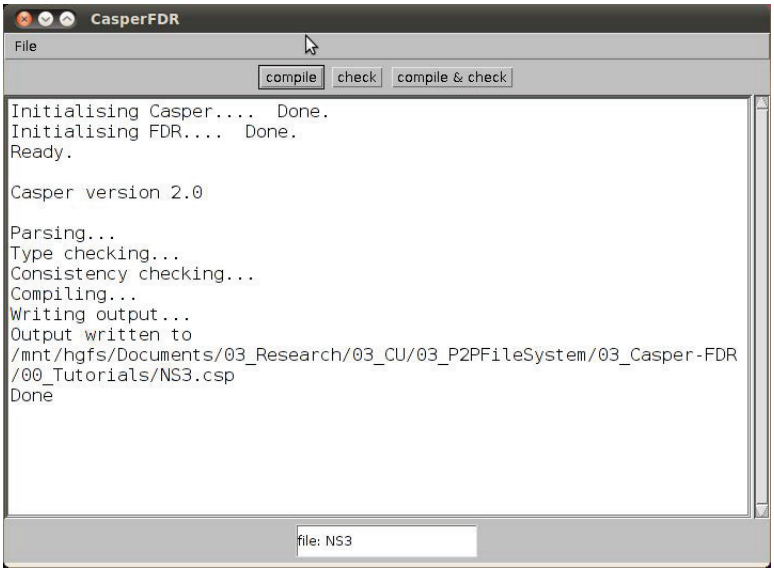


Figure 3.2: Compilation result of Needham-Schroeder protocol in Casper compiler

```
Intruder = Mallory
IntruderKnowledge = {Alice, Bob, Mallory, Nm, PK, SK(Mallory)}
```

3.1.2 Casper compiler

The Casper compiler runs over a Haskell compiler which generates functional programming languages from abstract languages. The Casper compiler generates CSP code from the Casper script which describe the protocols and verification model in abstract way. Figure 3.2 shows the compilation result of Needham-Schroeder protocol.

3.2 CSP

CSP (Communication Sequential Processes), proposed by C. A. R. Hoare in [28], is a process calculus for describing systems of multiple agents that communicate by passing messages. CSP can describe theoretical problems that arise from concurrency. In [49], P. Ryan and S. Schneider introduce how to formalize security properties by using CSP. In this paper, we do not model security protocols using CSP. We use Casper to model a security

protocol for security analysis since the CasperFDR tool can automatically generate CSP code from Casper script. Refer to [28] for modeling using CSP. We discuss basic CSP and introduce the similarity between CSP and Petri nets for performance analysis.

3.2.1 Basic building blocks of CSP

CSP has two classes of primitives; events, and processes. Events represent communications or interactions, and processes represent fundamental behaviors. The simplest example of CSP "Stop" shows how each primitive can be used.

$$in \rightarrow out \rightarrow Stop$$

in and *out* are events, and *Stop* is a process. It means that if input occurs, and output occurs then stop the procedure. Also, CSP provides various algebraic operators. In this paper, we introduce some, but not all, operators for security protocol modeling. Others are described in [28]. The prefix operator produces a new process from an event. For example, $a \rightarrow B$ produces process B when event a happens.

CSP provides choice operators. There are two different choice operators; deterministic and non-deterministic. In this paper, we use only non-deterministic choice. The nondeterministic choice operator, '$\sqcap$' represents a choice between two component processes, but does not allow the environment control over which component process will be selected. For example,

$$(a \rightarrow A) \sqcap (b \rightarrow B)$$

behaves like either $a \rightarrow A$ or $b \rightarrow B$.

The interleaving operator $\|$ represents independent concurrent activity. The process

$$A \parallel B$$

means both A and B occurs simultaneously.

CSP provides more, but we use only the operators introduced above. CSP descriptions of security protocols can be directly converted into Petri nets. A Petri net is a

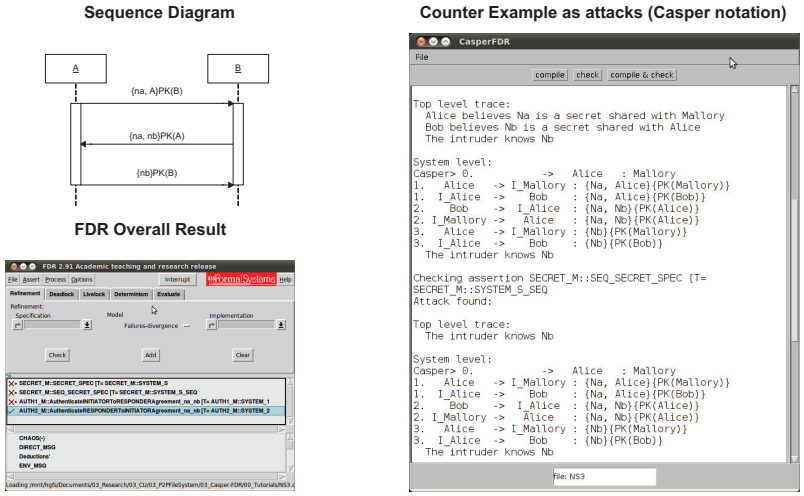


Figure 3.3: Needham-Schroeder protocol FDR test result

mathematical model for describing discrete distributed systems, which is an established tool for performance analysis [20]. The CSP operators used for security properties validation are similar to Petri nets. Details of Petri nets will be introduced in section 4.

3.3 FDR

FDR is an automated model checker [19] for analyzing a given security properties. A CSP model of the security protocol includes the message flows of the security protocol, the security property, and hostile intruders with explicit capacities, such as message dropping, message modification, etc. The CSP model is fed into FDR. The FDR result is that the security property is either fulfilled or not. If not, a counter example is returned.

We give an example analysis of the Needham-Schroeder protocol using Casper and FDR. Figure 3.3 shows the security analysis result of Needham-Schroeder protocol three message version.

The FDR result reveals that the 3 message Needham-Schroeder protocol has a secrecy vulnerability. The failure counter example follows.

Top level trace:

```
  Alice believes Na is a secret shared with Mallory
  Bob believes Nb is a secret shared with Alice
  The intruder knows Nb

System level:
Casper> 0.            ->    Alice    : Mallory
1.    Alice    -> I_Mallory : {Na, Alice}{PK(Mallory)}
1.   I_Alice   ->     Bob     : {Na, Alice}{PK(Bob)}
2.     Bob     ->   I_Alice   : {Na, Nb}{PK(Alice)}
2. I_Mallory   ->    Alice    : {Na, Nb}{PK(Alice)}
3.    Alice    -> I_Mallory : {Nb}{PK(Mallory)}
3.   I_Alice   ->     Bob     : {Nb}{PK(Bob)}
  The intruder knows Nb
```

The example shows that the intruder *Mallory* performs man-in-the-middle attack. He pretends to be *Bob* to *Allice* between them. Then he successfully steals the secret *nb*. Also, the authentication of both A to B, and B to A has failed. The following example reported by Casper shows how the intruder is taking the role of both *Alice* and *Bob*.

```
Checking assertion SECRET_M::SEQ_SECRET_SPEC
[T= SECRET_M::SYSTEM_S_SEQ
Attack found:

Top level trace:
  The intruder knows Nb

System level:
Casper> 0.            ->    Alice    : Mallory
1.    Alice    -> I_Mallory : {Na, Alice}{PK(Mallory)}
```

```
1.  I_Alice  ->     Bob     : {Na, Alice}{PK(Bob)}
2.     Bob   ->  I_Alice    : {Na, Nb}{PK(Alice)}
2. I_Mallory ->    Alice    : {Na, Nb}{PK(Alice)}
3.    Alice  -> I_Mallory   : {Nb}{PK(Mallory)}
3.  I_Alice  ->     Bob     : {Nb}{PK(Bob)}
  The intruder knows Nb

Checking assertion AUTH1_M::Authenticate
INITIATORToRESPONDERAgreement_na_nb [T= AUTH1_M::SYSTEM_1
Attack found:

Top level trace:
Bob believes he has completed a run of the protocol,
taking role RESPONDER, with Alice, using data items Na, Nb

System level:
Casper> 0.            ->    Alice    : Mallory
1.    Alice   -> I_Mallory  : {Na, Alice}{PK(Mallory)}
1.  I_Alice   ->     Bob    : {Na, Alice}{PK(Bob)}
2.     Bob    ->  I_Alice   : {Na, Nb}{PK(Alice)}
2. I_Mallory  ->    Alice   : {Na, Nb}{PK(Alice)}
3.    Alice   -> I_Mallory  : {Nb}{PK(Mallory)}
3.  I_Alice   ->     Bob    : {Nb}{PK(Bob)}
```

Chapter 4

Performance Evaluation

In this chapter, we evaluate the performance of security protocols by using Petri nets. Security protocols consist of cryptographic operations and communications among parties. The Petri net model includes cryptographic operations and communication the timing information. The model also includes system capacity issues, such as the number of users the system can support at a time. By evaluating the Petri net model, we obtain performance measures such as the average time to serve a user, the average time a user spends on cryptographic operations or communications, etc.

Karp and Miller [31] proposed a method to construct the reachability tree. This method guaranteed a finite reachability tree. The reachability graph is obtained by merging the same markings in the reachability tree. Karp Miller tree of a stochastic Petri net (SPN) is identical to the underlying Petri net model. By using the reachability tree, we can directly find which node is in deadlock, and which nodes are in livelock. The performance evaluation procedure is listed as follows:

1. Generate XML descriptions of security protocols from Casper-CSP models or sequence diagrams.

2. Generate Petri-Nets.

3. Generate Karp-Miller Trees.

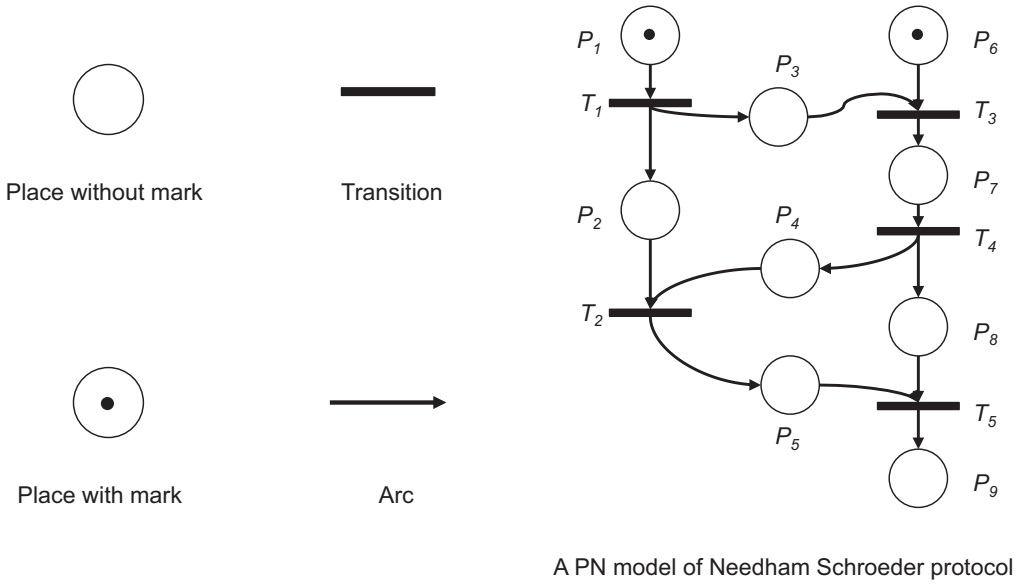


Figure 4.1: Basic components and an example of Petri net model

4. Checking livelocks and deadlocks.

5. Find bottlenecks.

4.1 Petri nets

Petri nets were invented by C. A. Petri, to model and visualize behaviors depicting parallelism, concurrency, synchronization, and resource sharing [20]. Petri net models are used for computer architecture and automatic control. In this paper, we apply a Petri net model to security protocols.

A Petri net consists of three components: places, transitions and arcs. Additionally, we add tokens (or marks). A Petri net model with marks on places is called a marked Petri net. Figure 4.1 shows basic components of Petri net model along with an example.

Bipartite graph with arcs between places (transitions) and transitions (places). The number of places and/or transitions should be neither infinite nor zero [20]. The example shown in Figure 4.1 is a Petri net representation of Needham-Schroeder consisting of 10

places and 5 transitions. Pictorially, a place is represented by a circle and a transition is represented by a box.

This paper uses the mathematical notation of the Petri net model introduced in [30]. Each place P_1 to P_7 has its own initial marking. A mathematical representation of Petri net is a 5-tuple $\{P, T, I, O, M_0\}$, where

- $P = \{P_1, P_2, ...\}$ is a finite set of places. For example, P of the example in Figure 4.1 is $P = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}\}$.
- $T = \{T_1, T_2, ...\}$ is a finite set of transitions disjoint from P. For example, T of the example in Figure 4.1 is $T = \{T_1, T_2, T_3, T_4, T_5\}$.
- I is a finite set of input arcs which satisfies $I_i \subseteq P \times T$.
- O is a finite set of input arcs which satisfies $O_i \subseteq T \times P$.
- M_0, initial marking is a non-negative vector of length $|P|$ with each element representing the number of tokens at each place P. For example, M_0 of the example in Figure 4.1 is $M_0 = [1\,0\,0\,0\,0\,0\,1\,0\,0\,0]^T$.

A place P_i is called input or output of transition T_j, and T_j is called an upstream or downstream transition of P_i when an arc $(T_j, P_i)((P_i, T_j))$ exists in a Petri net. For example, T_1 is an upstream transition of P_4, and T_3 is a downstream transition of P_4. If a transition does not have any downstream place, the transition is called to be a source transition. In opposition, if a transition has no upstream place, the transition is called to be a sink transition.

- $\prod(P_i) \subseteq T$: is a set of upstream transitions of place P_i.
- $\sigma(P_i) \subseteq T$: is a set of downstream transitions of place P_i.
- $\prod(T_j) \subseteq P$: is a set of input places of transition T_j,
- $\sigma(T_j) \subseteq P$: is a set of output places of transition T_j.

Table 1 Basic Event Structures Modeling in CSP and Petri Net

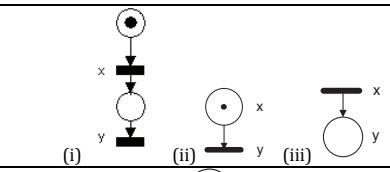
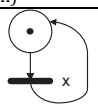
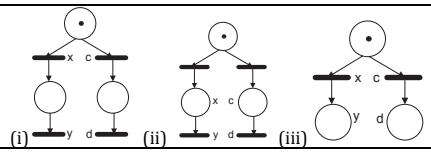
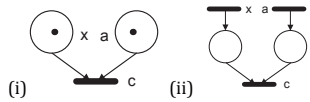
Event Structure	CSP model	Petri Net model
Sequential events	$x \to y$ (x and y are events from the same alphabet.)	
Recursion	$x \to x \to x\mathrm{L}$ (x is an event from an alphabet.)	
Choice	$x \to y \mid c \to d$ (x, y, c and d are events from the same alphabet.)	
Concurrency	$P \| Q$ (suppose processes are defined as $P : x \to c$ and $Q : a \to c$)	

Figure 4.2: Basic Event Structures Modeling in CSP and Petri Net [30]

4.2 Similarity between CSP and Petri net

P. Sheldon, J. Deng and R. R. Brooks observed the similarities between CSP and Petri net in [54, 23, 30]. The CSP operators shares many similarities with Petri net graphs. CSP supports modeling sequential events, events recursion, events choice and events concurrency, etc. Petri net can also support the same graph models. Figure 4.2 below shows the corresponding modeling of some basic event structures in CSP and Petri nets.

4.3 Reachability and Karp Miller Trees

A transition can be fired only if all of the input places for this transition contain at least one mark. Transition T_j is enabled by a Marking M if $M_i > w(P_i, T_j)$ for all $P_i \in \prod(T_i)$. Firing the enabled transition T_i generates a new marking $\bar{M}$

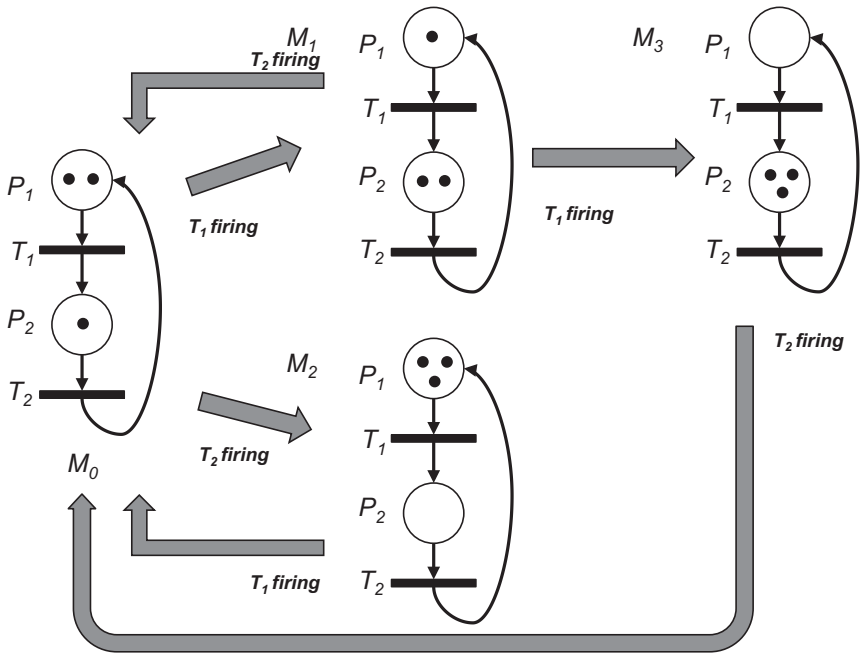


Figure 4.3: Transition firing examples of Petri net model

$$\bar{M} = \begin{cases} M_i - w(P_i, T_j), & \text{if } P_i \in \prod(T_j) \\ M_i + w(T_j, P_i), & \text{if } P_i \in \sigma(T_j) \\ M_i, & \text{otherwise} \end{cases}$$

$\bar{M}$ is reachable from M. Figure 4.3 shows an example of firing of transition by the firing rule mentioned above.

Places represent conditions and transitions represent events. Input places of a transition represent preconditions of the event, and Output places of a transition represent postconditions of the event. The presence of token(s) at place P_i represents that the precondition associated with P_i is fulfilled. At the initial state M_0, both P_1 and P_2 have more than one token. Thus, either T_1 or T_2 can be fired since the precondition of both T_1 and T_2 are fulfilled. As the result, state M_0 produces M_1 and M_2. At the state M_1, either T_1 or T_2 can be fired. If T_1 is fired, M_1 produces a new state M_3, but if T_2 is fired, the next state of M_1 becomes the same state as M_0. Thus, it goes to M_0 back. At the state M_2, only T_1 can

be fired, and the next state of M_1 becomes M_0. Unless otherwise stated, the probability of the transition each arc is 1.

The reachability of all nodes can be represented by a tree structure. The reachability tree of Petri Net $\{P, T, W, M_0\}$ is a tree with nodes obtained as:

1. M_0 is a node of this tree.
2. fire each enabled transition and obtain a new marking $\bar{M}$, and connect M_0 and $\bar{M}$ by an arc.
3. for each $\bar{M}$, recursively perform step 2.

The reachability tree of Figure 4.3 is shown in Figure 4.4. The example has only finite reachability, the tree has the exact same structure of Figure 4.3. The tree covers all possible states of the Petri net model. Each state M_i has a set of markings of all places:

- $M_0 = \{2, 1\}$
- $M_1 = \{1, 2\}$
- $M_2 = \{3, 0\}$
- $M_3 = \{0, 3\}$

If a Petri net is not bounded, the reachability graph of the Petri net also has an infinite number of nodes. The reachability tree constructed by the above procedures may become infinite for some Petri nets, which make the analysis extremely difficult. Karp Miller tree guarantees a finite reachability although the reachability of some nodes goes to infinity. The Karp Millet tree is obtained by merging the same markings in the reachability tree. To guarantee the finiteness of rechability graph, Karp Miller trees use ω to represent an infinite number of tokens. The algorithm to produce finite nodes of Karp Miller tree using ω is as follows:

1. M_0 is a node of this tree.

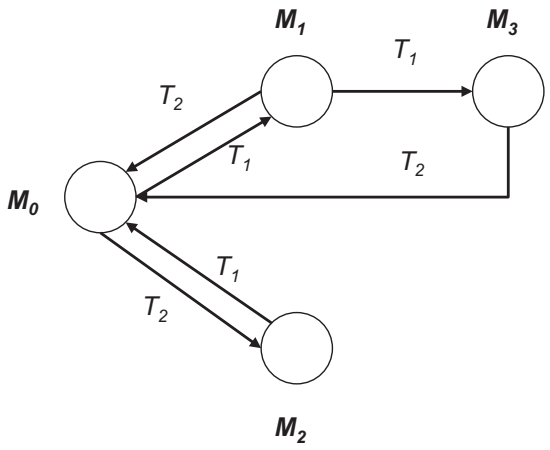


Figure 4.4: The reachability tree of the Petri net example

2. Fire each enabled transition and obtain a new marking $\bar{M}$, and connect M_0 and $\bar{M}$ by an arc.

3. If $\bar{M}_i$ covers M_i ($\bar{M}_i > M_i$), then substitute the marking of $\bar{M}_i$ which covers the marking of M_i to ω.

4. If the markings of $\bar{M}_i$ already exists as M_j in the reachability tree, merge $\bar{M}_i$ with M_j.

5. For each $\bar{M}$, recursively perform step 2.

4.4 Connectivity Matrix

One of our objectives in performance tests is finding deadlocks and livelocks of the security tools. These performance problems can be detected by using the Petri net of communication protocol models. To find the problems, we need to define connectivity matrix of Karp Miller tree which shows all possible next states from the current state. In the previous section we confirm that if the Perti net has a finite length, Karp Miller tree always

have finite node length. From the finite length Karp Miller tree, we derive a connectivity matrix. The connectivity matrix W can be defined as a $n \times n$ square matrix, such as:

$$W = \begin{bmatrix} [r]a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{1,1} & a_{1,1} & \cdots & a_{1,n} \\ a_{3,1} & a_{1,1} & a_{1,1} & \cdots & a_{1,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,1} & a_{n,1} & \cdots & a_{n,n} \end{bmatrix}$$

When a Karp Miller tree has n nodes of . By using this connectivity matrix, we can find the next state of Karp Miller trees. The current state of the Karp Miller tree is defined as $M_i = [m_1\, m_2\, m_3 \cdots m_n]$ where,

$m_k = 1$ only if current state is at k-th node, otherwise $m_k = 0$. $(0 < k \leq n\ )$.

The next state M_{i+1} is:

$M_{i+1} = M_i \times W$

$$[m_{1,1}\, m_{1,2}\, m_{1,3} \cdots m_{1,n}]^+ = [m_{1,1}\, m_{1,2}\, m_{1,3} \cdots m_{1,n}] \begin{bmatrix} [r]w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{1,1} & w_{1,1} & \cdots & w_{1,n} \\ w_{3,1} & w_{1,1} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,1} & w_{n,1} & \cdots & w_{n,n} \end{bmatrix}$$

M_i of the current state only has one 1 but M_{i+1} can have multiple of 1s because the vector shows the possibility of next states.

Then, we can find the possibility of $M_(i + k)$ by multiplying k-times of W.

$M_{i+k} = M_i \times W^k$

$0 < k \leq n$ The maximum of k is the length of the Karp Miller tree's node n because the longest path of node transitions is to pass all nodes.

4.5 Deadlocks and Livelocks

We can find deadlocks and livelock from:

$\sum_{k=1}^{n} W^k = W^1 + W^2 + W^3 + \cdots + W^k$

If a row j of $\sum_{k=1}^{n} W^k$ has all '0' where $0 < k \leq n$, it means that there is no possibility to reach the node j. Therefore, the node j is a deadlock node. Livelocks can also be found from $\sum_{k=1}^{n} W^k$. If an $w_{i,j} \neq w_{j,i}$ they are connected components, and $w_{i,j}$ and $w_{j,i}$are livelocks.

4.6 Bottleneck Analysis

A deterministic and stochastic Petri net (DSPN) may be seen as a seven-tuple of $\{P, T_D, T_S, W, M_0, \beta, \Lambda\}$. $\{P, T, W, M_0\}$ is the underlying Petri net with $T = T_D \cup T_S$. There are two sets of transitions in a DSPN. T_D is a set of transitions with determined firing times specified in β and T_s is the set of transitions with variable firing times, which follow exponential distribution with rates specified in λ. The firing rates may either be marking-dependent or marking-independent. The average firing time of firing time of transition $T_S j$ is $1/\lambda_j$. Pictorially, transitions with constant firing times are drawn as a black boxes and others are drawn as white boxes. The reachability graph of a stochastic Petri net is identical to the underlying Petri net.

Chapter 5

Distributed Authentication and Authorization

Parallel distributed systems mostly deal with important data. The security requirements of the systems should be different from centralized systems since the centralized security tools are likely to cause bottlenecks and have a possibility of single point failure [10]. Many distributed security tools have been developed to keep the system from security threats; Web of Trust [11], X.509 Public Key Infrastructure[8], WS-SecurityKerberos[40], and so on[37, 26, 29].

The security tools help the system maintain important security attributes. The fundamental security issues of parallel distributed systems are authentication and authorization; participants are authenticated; access rights are verified at session initiation and revoked as necessary [10].

In this chapter, we introduce Kerberos , PERMIS, and Shibboleth. We observe cryptography protocols and communication procedures of each security tool, getting information to build the sequence diagram as the starting point of security and performance verification.

5.1 Kerberos

Kerberos is a distributed authentication system, which was developed for the project Athena workstation system to manage authentication of a distributed system[39]. This authentication model relies on symmetric key exchange through the trusted third parties[41]. Key Distribution Center (KDC) which consists of authentication server (AS) and ticket granting server (TGS), only knows the secret keys of all and issues tickets between its servers via the client who wants to be authenticated[43, 39]. Since Kerberos can be configured with multiple TGSs, and service servers, the system can support multiple realm authentication.

The basic assumption of the Kerberos is that session keys and the key management servers must not be compromised. This ticket-based authentication system enables a client to prove its identity without knowing the cryptography information of target server. Standard configuration of Kerberos uses two key management servers: AS and TGS. Also, two independent tickets are used: TGS ticket and service ticket. The authentication process is done by exchange tickets among a principal, KDC and service server (SS). Figure 5.1 shows how the authentication process of Kerberos works.

The authentication procedure is as follows [43]:

1. As the first step, a user on the client enters the username and password on the client. The client sends the log on authentication request with user name and encrypted password to AS.

2. AS checks the password and log on information. If the log on information given by the user is correct, then AS issues TGS ticket which is encrypted with the user's password.

3. The client decrypts and verifies the ticket received. The ticket includes TGS ticket but the TGS ticket is encrypted using the secret key of TGS. Thus, the client only can get the client/TGS session key, and the server's identity. The client sends the TGS ticket and authenticator to TGS. The authenticator is the message which only contains the identity of client, and timestamp.

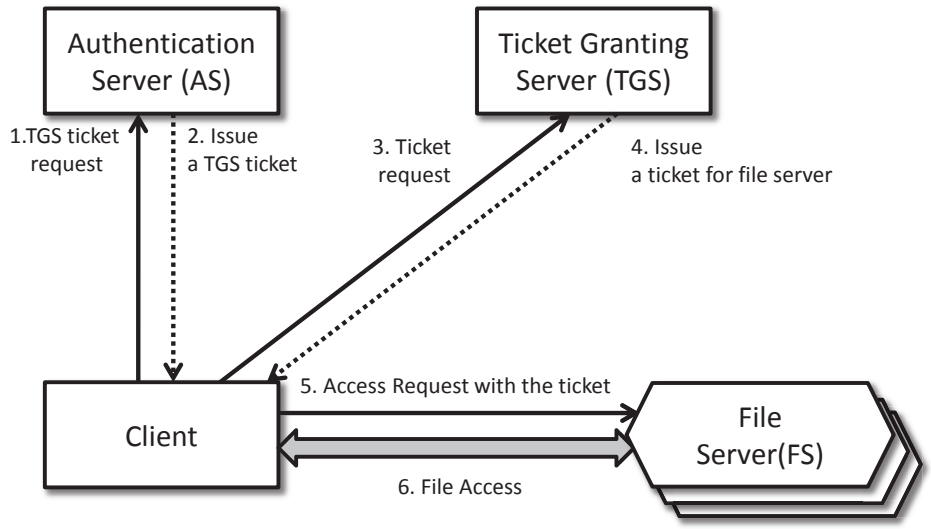


Figure 5.1: The system diagram of Kerberos

4. The TGS decrypts and verifies the TGS ticket and the authenticator. If all the information is correct, TGS issues service a ticket which includes client/SS session key. The ticket is encrypted with the secret key of the SS.

5. The client decrypts and verifies the service ticket. If it is correct, the client sends the ticket and an newly generated authenticator encrypted with client/TGS session key to SS.

6. The SS decrypts and verifies the ticket and the authenticator. If all information is correct, the SS release client/SS session key encrypted with the client/TGS session key.

Using symmetric keys, Kerberos has more advantages on cryptography speed and less computing power over asymmetric key authentication services. Also, current version of Kerberos can support PKI for session key exchange and ticket forwarding to allow the request from non Kerberos server which is under the control of authenticated Kerberos client. Configuring Kerberos with multiple domain and ticket delegation also increases

distributed authentication performance, but the fact that Kerberos only allows a single AS can cause traffic bottleneck.

5.2 PERMIS

PERMIS (PrivilEge and Role Management Infrastructure Standard), developed by D. W. Chadwick [15], is a distributed authorization system which supports assignment of both policy and attributes to users. PERMIS supports hierarchical role based access controls in which policies are organized in a hierarchy and inheritance of policies.

PERMIS provides a Privilege Management Infrastructure (PMI) based on public key cryptography and X.509 attribute certificates (ACs) to manage user attributes [14]. PERMIS does not provide any authentication mechanism, but the system can be integrated with other existing authentication systems. Since PERMIS supports WS-Security, most types of security tokens defined by WS-Security specification can be embedded into PERMIS ACs; such as X.509 certificates, Kerberos, UserID/password credentials, and SAML assertions [60].

A PERMIS authorization service is configured with many individual components. This makes PERMIS look complicated, but it helps to achieve hierarchical, strict and precise access controls. The individual components of PERMIS infrastructure are:

- Policy enforcement point (PEP) is a gateway for the authorization. Subject PEP collects user certificates, attributes, and access control decision, or sends the access control decision to object server. The communication for cross domain authorization can be done by multiple PEPs.

- Policy decision point (PDP) makes authorization decisions based on the attributes and access control policies. Decision making process contains two categories of rules, trust related rules (Credential Validation Policy) and privilege related rules (Access Control Policy). PDP informs PEP which policies to include with the user request.

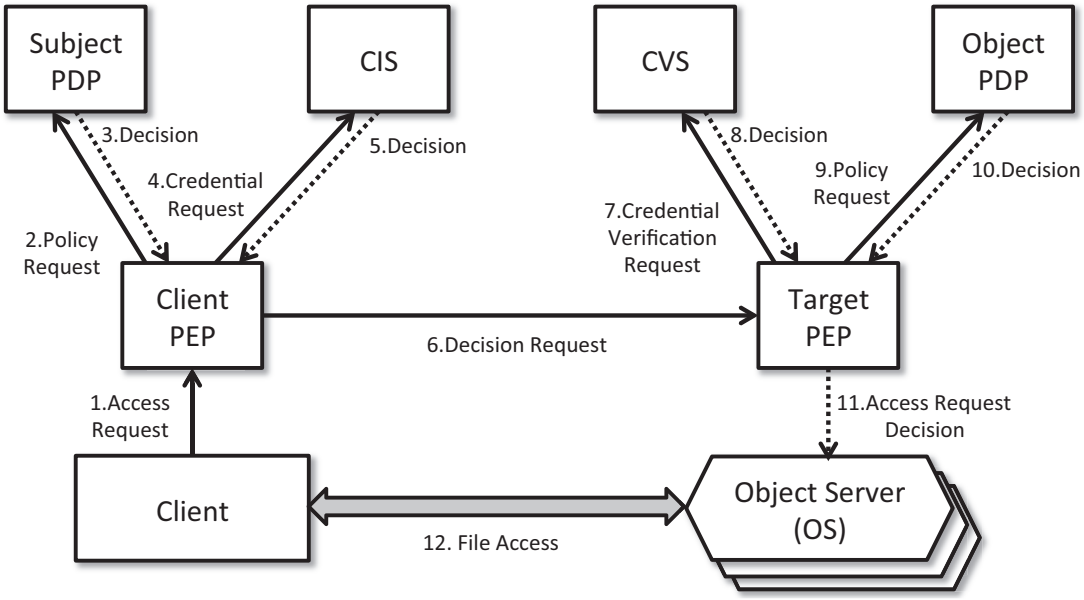


Figure 5.2: The system diagram of PERMIS

- Credential issuing service(CIS) verifies user information and issues certificates when the user provides CIS with correct identity through client PEP.
- Credential validation service(CVS) verifies the user credentials received from target PEP. CVS informs PEP which credentials from which issuing attribute authorities (AA) are trusted by the object server. AA is an entity trusted by one or more entities to create and sign ACs. A certification authority(CA) may also be an AA [5].
- Credentials and policies repository (CPR) uses LDAP as a network accessible repository to store and manage policies including X.509 ACs. CPR sends the ACs to PDP when they need to make authorization decision [12].

Figure 5.2 shows how PERMIS authorization process works over multiple domains. The details of the authorization process are as follow:

1. The protocol starts from sending an access request from client user to client PEP. The access request includes a set of attributes issued by AA.
2. The PEP send the ACs to PDP.

3. PDP verifies the ACs. If the ACs are validated, the PDP informs client PEP which policies to include with the user request.

4. Client PEP requests a new certificate to CIS.

5. CIS issues a new AC which includes the attributes of user and target resources.

6. Client PEP sends the user's request with new ACs to target PEP.

7. Target PEP sends the ACs received from client PEP to CVS for credential validation.

8. CVS returns the validation result.

9. If the ACs are validated, PEP send the validated attributes along with date and time stamp to the PDP for access control decision.

10. PDP returns access control decision.

11. If the decision is granted, the PEP allows the user access, otherwise the access is rejected.

To manage complicated policies, PERMIS infrastructure provides GUI policy editor for system administrator to easily compose and edit policies. Policies are written in XML format with embedded X.509 ACs and stored into LDAP directories of CPR. PERMIS also provides the attribute certificate manager(ACM) tool which allows administrators to create new ACs or assign attributes to users. Administrator also can modify or revoke existing ACs in LDAP repositories.

5.3 Shibboleth

Shibboleth is an integration solution for federated authentication and authorization, developed as a middleware of Internet2 project. Similar to PERMIS, Shibboleth does not provide any internal authentication service [42]. The objective of Shibboleth is achieving single sign on and exchange of the authentication and authorization information among

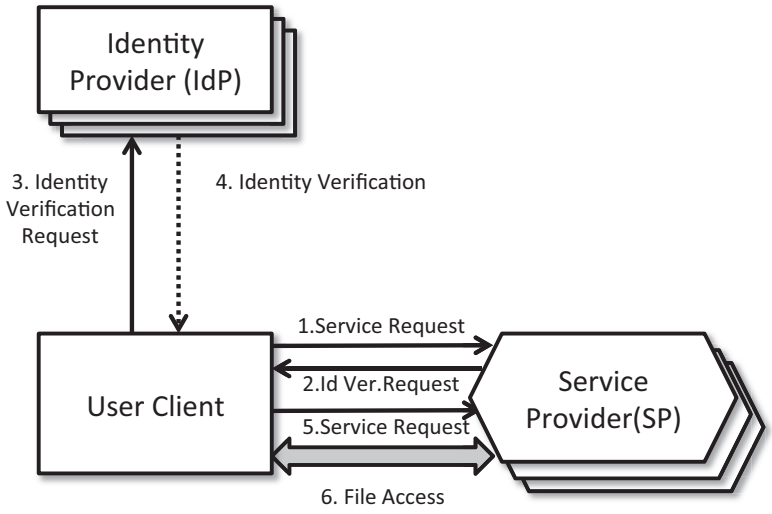


Figure 5.3: The system diagram of Shibboleth

the federated institutions[24]. To achieve this goal, Shibboleth provides related functionalities such as federated administration, heterogeneous authentication systems, access control based on attributes, and a strong emphasis on user-managed privacy [27].

The security protocols of Shibboleth relies on security assertion markup language (SAML) for the communication security and data management[42]. SAML is a security extension of extensible markup language (XML). In 2005, SAML 2.0 became an data exchange standard of organization for the advancement of structured information standards (OASIS). SAML 2.0 supports both XML encryption and XML signature [4]. Shibboleth uses only XML signature for SAML assertions and SAML profile to describe specific use scenarios [27]. A simple configuration of Shibboleth includes an Identity provider (IdP) which creates, manages, and verifies user certificates and attributes, and a service provider (SP) which manages the restricted service. In order for Shibboleth to work, the IdP and SP must trust each other [36]. Figure 5.3 shows the configuration of Shibboleth.

The authentication sequence is as follows:

1. Client sends a request and user attributes to access the restricted service provided by

SP.

2. SP tries to find the certificates of the client. If the certificate does not exist in SP. SP redirect the service request with attributes verification request to IdP.

3. Client sends the request packet to IdP.

4. IdP verifies the request. If the user attributes are valid, IdP redirects the service request with verification result to SP.

5. Client sends send the request and verification results.

6. SP allows the client the restricted service.

Chapter 6

Implementation

In this chapter, we apply the security verification and performance evaluation methodologies to Kerberos, PERMIS, and Shibboleth to find security vulnerability and performance bottlenecks. To apply the test methodologies, we should clarify the cryptography methods and communication processes of each security tool.

The CSP operators shares many similarities with PN graphs. CSP supports modeling of sequential events, events recursion, events choice and events concurrency, etc. PN also can support same graph models. However, modeling security protocols using CSP is too complicated and prone to error as well as, converting from CSP to PN creates enormous graphs to analyze the security protocol.

In chapter 3, we confirmed that sequence diagram can directly be converted into Casper scripts. We also observed the similarities between sequence diagram and PN. Converting sequence diagram directly to Casper scripts and PN graphs overcomes these disadvantages. Figure 6.3 shows the overall process of security and performance verification.

6.1 UML sequence diagram

We use sequence diagram of Unified Modeling Language (UML) 2.0 to clarify the security and communication process specification. UML was developed for designing and de-

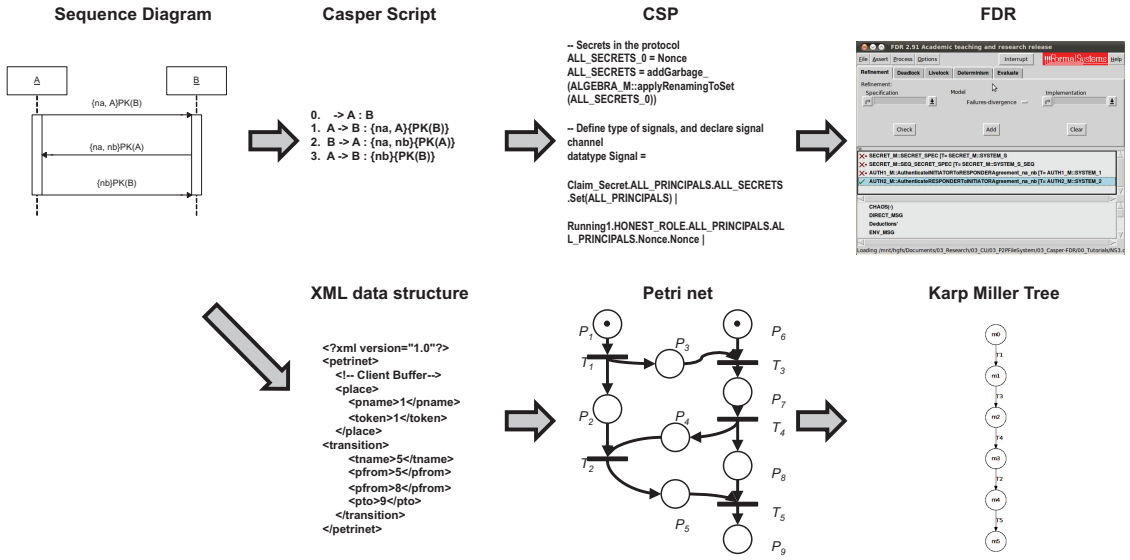


Figure 6.1: Overall process of security verification and performance evaluation

Notation	***Description***
t_x	timestamp of entity x
K_x	encryption key of entity x
$K_{x,y}$	Session key between entity x and y
$\{M\}K_x$	message M encrypted with key K_x

Table 6.1: Mathematical Notation for Security Protocols

scribing object-oriented program in software engineering. The sequence diagram of UML is designed for modeling the interactions between objects or entities in the sequential order[7]. [44] shows how to use UML for designing a security model. Using the sequence diagram we can describe both the communication processes and cryptography information of security tools in a diagram. To represent security protocol using UML sequence diagram, we use mathematical notation as shown in Table 6.1.

Several types of sequence diagrams exist, but we only use system level sequence diagrams to describe cryptography information. More information on various sequence diagram and how the system level sequence diagrams are different from service level sequence diagrams can be found from [6].

The rest of the chapter is organized as follows: Section 6.2 shows the security analyzation of each distributed authentication and authorization tool. The section mainly explains sequence diagrams, Casper scripts of Kerberos, PERMIS, and Shibboleth. The FDR results of the security protocols are dealt with in chapter 7. Section 6.3 shows how to write the XML data structure designed for describing security protocols. It also explains the design of the performance analysis program which generates PN graphs and Karp Miller tree graphs from the XML data structure scripts. This section shows the PN graphs and Karp Miller trees of Kerberos, PERMIS, and Shibboleth. The results of locating deadlocks, livelocks, and bottlenecks of each protocol are shown in chapter 7.

6.2 Security verification

System level sequence diagram of UML can simply depict security protocols including the information of communication process and cryptography information. Casper also can precisely describe security protocols in abstract language, however writing casper scripts without their sequence diagrams is prone to make errors. To reduce implementation errors, we draw sequence diagram first. In section 6.3, we will use the sequence diagrams to write XML protocol descriptions for each protocol.

6.2.1 Kerberos

In section 5.1, we summarized the service level protocol information of Kerberos, but we need more information to describe the detail of cryptography information. The cryptography information of real Kerberos protocol can be found in [56]. First we define the server names used for the sequence diagram in table 6.2.

Kerberos uses two types of credentials; a ticket and an authenticator. Both are used for the authentication between two entities. The difference between the ticket and the authenticator is the lifetime. The ticket can be repeatedly used until the ticket expires, but the authenticator can only be used once. The ticket contains both entity names, IP address

Notation	*Description*
C	client name
AS	authentication server name
TGS	TGS server name
SS	service server name

Table 6.2: Mathematical Notation used for Kerberos

of the client, a timestamp, a lifetime, and a random session key between the two entities. In this paper, we do not use the IP address of the client, and lifetime of the key. This ticket is encrypted with the key of destination entity. In Kerberos system, since the user of client has only its own key, the user can pass the ticket but cannot decrypt the ticket. A ticket $T_{x,y}$ is for y to grant access control to x can be defined as follows;

$$T_{x,y} = \{x, y, t_x, K_{x,y}\}K_y$$

x is the name of source entity, and y is the name of destination entity. t_x represents the timestamp of the ticket issuing time, and $K_{x,y}$ is the random session key between x and y. The ticket is encrypted with the secret key of y. The message structures of Kerberos authenticator A_x can be defined as follows:

$$A_{x,y} = \{x, t_x\}K_{x,y}$$

The authenticator should be encrypted with the session key between x and y. In this paper, our authenticator includes only the name of source entity and the timestamp. The authenticator is encrypted by the session key. Figure 6.2 shows the system level sequence diagram of Kerberos adding the ticket and the authenticator information to the diagram shown in Figure 5.1 of Chapter 5. The sequence diagram includes both communication processes and cryptography messages. The communication procedure was described in section 5.1.

As mentioned above, the sequence diagram can directly be translated into Casper script. One problem is current version of Casper cannot compile a model with more than

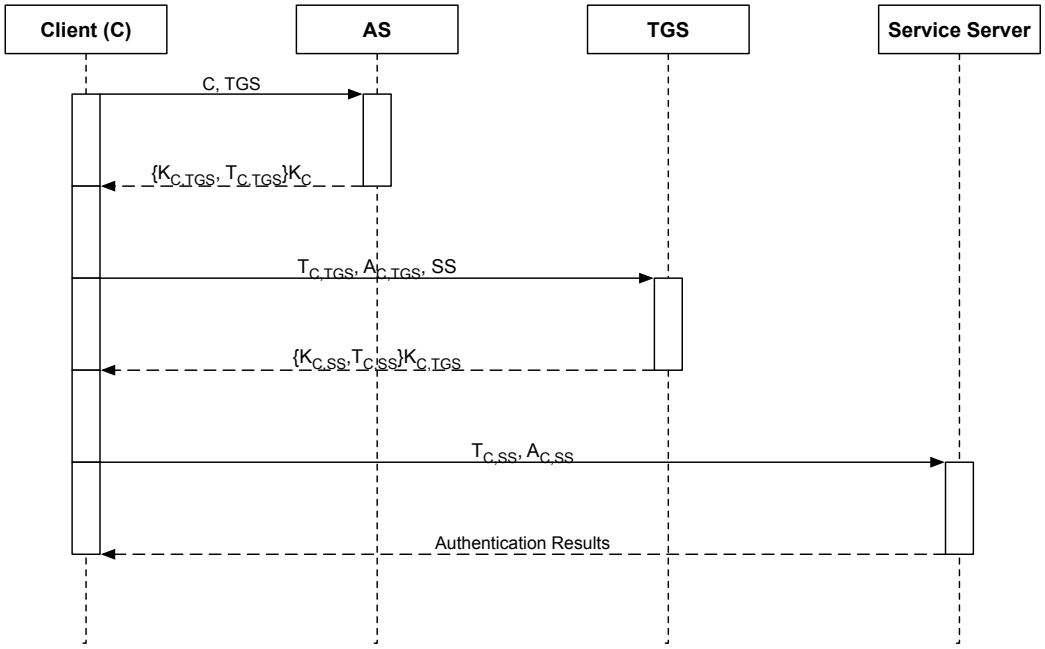


Figure 6.2: System level sequence diagram of Kerberos

three servers. We therefore break the communication protocol into several parts with overlapping. Kerberos can be divided into two parts. The full Casper scripts of Kerberos are shown in Appendix D.

6.2.2 PERMIS

The service level descriptions was explained in section 5.2. We add cryptography information of PERMIS into the sequence diagram to build the security verification model. The cryptography information of PERMIS protocol can be found from [13, 21, 59, 22]. PERMIS uses two cryptography techniques; WS-TRUST security token and secure hash function. WS-TRUST is used for credential validation of subject. and encrypted by public key of CVS. Secure hash is used for digital signature of SAML attribute assertion. The structure of SAML attribute structure is shown in Table 6.3.

The issuer field is optional. The entity name could be subject name or target object name. Since PERMIS uses Simple Public Key Infrastructure (SPKI) based on digital

Subject or Object	Attribute ID	Value	Data Type	Issuer

Table 6.3: The Structure of SAML Attribute Assertion in PERMIS [22]

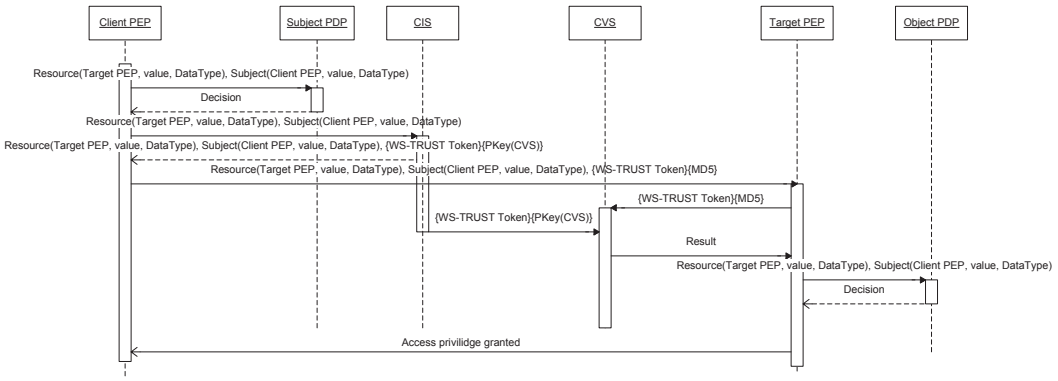


Figure 6.3: System level sequence diagram of PERMIS

signature technology, the public key signature encrypted by secure hash function comes with the attributes [14].

Converting the sequence diagram of PERMIS into Casper scripts need more techniques than used in Kerberos. The Casper scripts of PERMIS require the implementation of WS-TRUST token, and MD5 secure hash. We substitute the WS-TRUST token with simple public key encryption, and MD5 hash with internal hash function. The communication protocol of PERMIS is divided into three parts and implemented. Figure 6.3 is the system level sequence diagram of PERMIS based on the security information. The full Casper scripts of PERMIS can be found in Appendix D.

6.2.3 Shibboleth

The service level description was explained in section 5.3. Shibboleth has only two service entities; Identity provider (IdP) and Service provider (SP). The communication message protocol and security is done by using SAML assertions. Shibboleth We add cryptography information of Shibboleth into the sequence diagram to build the security

EL	**samlp:Request**	MUST appear once and only once
ATT	**RequestID**	
ATT	**MajorVersion**	MUST equal "1"
ATT	**MinorVersion**	MUST equal "0"
ATT	**IssueInstant**	MUST equal the current GMT date and time
EL	**samlp:RespondWith***	MAY appear zero or more times MUST connote a SAML AttributeStatement, if used
EL	**ds:Signature?**	MAY contain an XML signature MAY include an X.509 certificate
EL	**samlp:AttributeQuery**	
ATT	**Resource**	SHOULD contain the target URL, if applicable
EL	**saml:Subjectt**	Subject information

Table 6.4: Attribute Query Message (AQM) Common Syntax [24]

verification model. The document [24] written by M. Erdos contains detailed information on Shibboleth communication protocol and message structure. Shibboleth uses two different types of messages; attribute query message(AQM) and attribute response message(ARM). All Shibboleth protocols must share the AQM and ARM. The structure of AQM and ARM attribute structure is shown in table 6.4 and 6.5.

We only focus on the signature fields. The signature field contains X.509 certificate or XML signature. We implement the signatures using public key signing. Figure 3 is the system level sequence diagram of PERMIS based on the security information. The full Casper scripts of Shibboleth can be found in Appendix D.

6.3 Performance Evaluation

For the performance test, we developed a test tool named "Performance Evaluation Tools for Security protocols" (PETS). The program can generate Petri net graphs from XML data structures and convert it into Karp Miller trees. The program can calculate the number of deadlocks and livelocks of the Karp Miller Tree.

PETS software consists of three classes: [PETS], [PetriNet], and [KManalysis]

EL	**samlp:Request**	MUST appear once and only once
ATT	**RequestID**	
ATT	**InResponseTo**	MUST equal RequestID in AQM
ATT	**MajorVersion**	MUST equal “1”
ATT	**MinorVersion**	MUST equal “0”
ATT	**IssueInstant**	MUST equal the current GMT date and time
EL	**ds:Signature?**	MAY contain an XML signature MAY include an X.509 certificate
EL	**saml:Assertion**	
ATT	**AssertionID**	
ATT	**MajorVersion**	MUST equal “1”
ATT	**MinorVersion**	MUST equal “0”
ATT	**Issuer**	
ATT	**IssueInstant**	SHOULD equal GMT date and time of statement generation
EL	**saml:AttributeStatement**	MUST appear once and only once
EL	**saml:Subject**	Subject information
EL	**saml:Attribute***	Subject information
EL	**Isaml:Conditions**	
ATT	**NotBefore**	SHOULD be omitted, MUST be in the past
ATT	**NotOnOrAfter**	MAY be used to signify attribute expiration
EL	**ds:Signature?**	MAY contain an XML signature MAY include an X.509 certificate

Table 6.5: Attribute Response Message (ARM) Common Syntax [24]

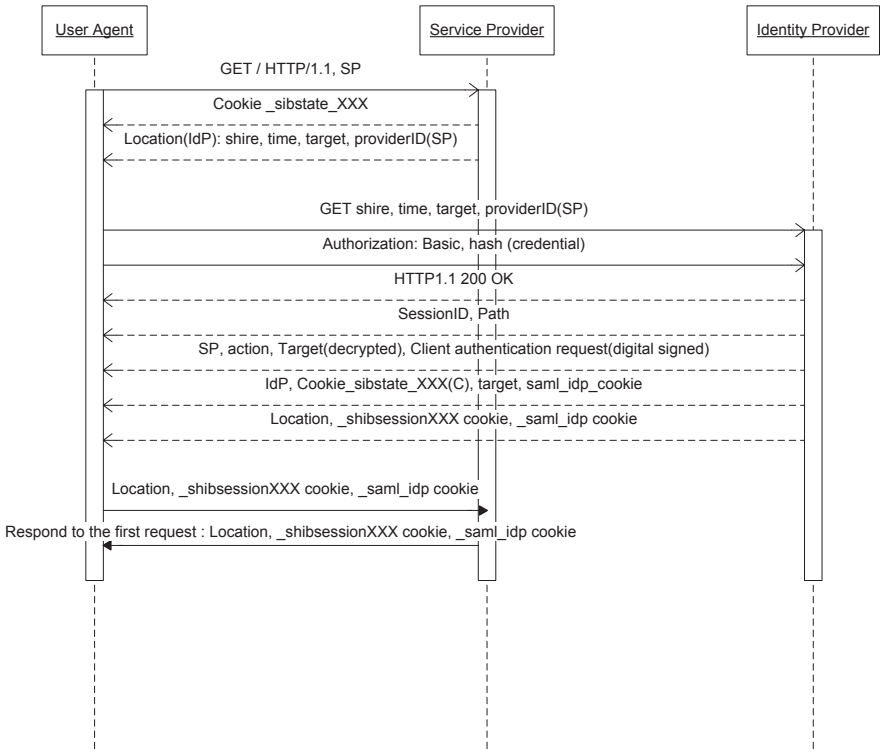


Figure 6.4: System level sequence diagram of Shibboleth

classes. [PETS] class is the main class. The class has the functionalities as follow:

- Read petrinet XML data structures.
- Draw petrinet graphs from XML data structure.
- Generate Karph Miller trees from the petrinets using [KManalysis] class.
- Draw Karph Miller trees using [PetriNet] class.

The Petri net graphs and Karp Miller trees of each protocol can be found in Appendix B and C. The rest of the chapter shows the techniques that we used for evaluating the performance of security tools.

6.3.1 XML data structure for protocol description

XML data structure makes it easier to design a complicated data base model. The tagging style of XML looks similar to HTML, but XML allows any type of tags unlike HTML [38]. In this paper, we define a data structure of XML for describing Petri net graphs.

The data structure begins with $< petrinet >$ and end with $< /petrinet >$. The petrinet structure include two sections, $< place > \cdots < /place >$ are for defining place nodes of Petri nets, $< transition > \cdots < /transition >$ are for transition nodes. The place section includes $< token > \cdots < /token >$ which indicates that the number tokens the place has. Each transition node of the the transition section has arc definitions. There are two types of arcs: upstream and downstream. Upstream arcs are defined by $< pfrom > \cdots < /pfrom >$, and downstream arcs are defined by $< pto > \cdots < /pto >$. The following code is a part of Needham-Schroeder protocol that we have used as an example.

```
<?xml version="1.0"?>
<!-- Place definition -->
<petrinet>
    <!-- Client Node -->
```

```
    <place>
        <pname>1</pname>
        <token>1</token>
    </place>
    <place>
        <pname>2</pname>
        <token>0</token>
    </place>
<!-- Transition definition -->
    <transition>
        <tname>1</tname>
        <pfrom>1</pfrom>
        <pto>2</pto>
        <pto>3</pto>
    </transition>
    <transition>
        <tname>2</tname>
        <pfrom>2</pfrom>
        <pfrom>4</pfrom>
        <pto>5</pto>
    </transition>
</petrinet>
```

Chapter 7

Test Results

By using the methodologies that we introduced in previous chapters, we verify the security and performance of Kerberos, PERMIS, and Shibboleth. The purpose of the security test is to ensure the authentication between the client and every service entity. The purpose of the performance test is finding deadlocks and livelocks in each protocol.

7.1 Security test result

We inject possible malicious attacks in the Casper scripts. The table 7.1, 7.2, and 7.3 shows summary of the attacks and test results. From the test, we found no security vulnerability on the protocols against the attacks we injected. Detailed results of each security tool can be found in appendix A.

7.1.1 Performance Test Results

We have checked whether Kerberos, PERMIS, or Shibboleth has any livelock or deadlock on its protocol by analyzing each Petri net. The Petri net graph of each security tool can be found in appendix B. Also, Karp Miller trees of Kerberos, PERMIS, and Shibboleth generated with two initial tokens can be found in appendix C. Since we cannot perform bottleneck tests with current test tool, we cannot define the number of initial tokens in the

Table 7.1: Security Test result of Kerberos

Protocol	**Test Type**	**Test Spec.**	**Result**
Kerberos	Timed Agreement Test	TimedAgreement(C, TGS, 2, [])	OK
		TimedAgreement(C, DS, 2, [])	OK
	Agreement Test	Agreement(C, AS, [])	OK
		Agreement(AS, C, [])	OK
		Agreement(TGS, C, [])	OK

C: *Client*, AS: *Authentication Server*, TGS: *Ticket Granting Server*

Table 7.2: Security Test result of PERMIS

Protocol	**Test Type**	**Test Spec.**	**Result**
PERMIS	Secrecy Test	Secret(SPDP,rsc,[CIS, CPEP])	OK
		Secret(CIS,wst,[SPDP, CPEP])	OK
		Secret(CPEP, rsc, [TPEP])	OK
		Secret(CPEP, wst, [CVS])	OK
		Secret(OPDP, rsc, [TPEP, CPEP])	OK
		Secret(OPDP, na, [CPEP, TPEP])	OK
	Agreement Test	Agreement(SPDP, CIS,[])	OK

CPEP: *Client Policy Enforcement Point*, SPDP: *Subject Policy Decision Point*
TPEP: *Target Policy Enforcement Point*, OPDP: *Object Policy Decision Point*
CIS: *Credential Issuing Server*, CVS: *Credential Verification Server*

Table 7.3: Security Test result of Shibboleth

Protocol	**Test Type**	**Test Spec.**	**Result**
Shibboleth	Secrecy Test	Secret(SP,na,[IP])	OK
		Secret(IP,nb,[SP])	OK
	Agreement Test	Agreement(SP, IP,[])	OK
		Agreement(IP, SP,[])	OK

SP: *Service Provider*, IP: *Identity Provider*

worst condition. Therefore, we have performed the deadlock and livelock test five times for each protocol with different number of initial tokens. The table 7.4 shows the summary of

the test results.

Table 7.4: Livelock and Deadlock Test Results

Protocol	**Test Type**	**Token 1**	**2**	**3**	**4**	**5**
Kerberos	Livelock	No	No	No	No	No
	Deadlock	No	No	No	No	No
PERMIS	Livelock	No	No	No	No	No
	Deadlock	No	No	No	No	No
Shibboleth	Livelock	No	No	No	No	No
	Deadlock	No	No	No	No	No

Chapter 8

Conclusions and Future Work

We have introduced distributed authentication and authorization protocols for the security of distributed file systems. We examined details of the background knowledge on test tools and various distributed authentication and authorization tools.

We showed how Casper, CSP, and FDR can be used for the security verification of distributed authentication and authorization tools. We newly implemented the performance verification software based on Petri nets and Karp Miller tree analysis, and showed the security and performance test results of Kerberos, Shibboleth, and PERMIS by using the software tools. From the test results we found no security issue nor performance lack. However, we haven't finished bottleneck test yet since the test requires measuring real execution time of each protocol.

The next step of this study is finding bottlenecks of each security tool, and apply the test methodologies to more distributed authentication and authorization tools.

Appendices

Appendix A Security Test Results on FDR

A.1 Kerberos

```
Starting FDR
Checking /mnt/hgfs/Documents/03_Research/03_CU/03_P2PFileSystem/03_Casper-FDR
    /01_Kerberos/Kerberos1of2.csp

Checking assertion AUTH1_M::AuthenticateINITIATORToSERVERAgreement [T= AUTH1_M
    ::SYSTEM_1
No attack found

Checking assertion AUTH2_M::AuthenticateSERVERToINITIATORAgreement [T= AUTH2_M
    ::SYSTEM_2
No attack found

Checking assertion AUTH3_M::AuthenticateINITIATORToRESPONDERTimedAgreement2 [T
    = AUTH3_M::SYSTEM_3
No attack found

Checking assertion STOP [T= SYSTEM\diff(Events,{|INTRUDER_M::verify|})
No attack found

Done

Starting FDR
Checking /mnt/hgfs/Documents/03_Research/03_CU/03_P2PFileSystem/03_Casper-FDR
    /01_Kerberos/Kerberos2of2.csp

Checking assertion AUTH1_M::AuthenticateSERVERToINITIATORAgreement [T= AUTH1_M
    ::SYSTEM_1
No attack found

Checking assertion AUTH2_M::AuthenticateINITIATORToRESPONDERTimedAgreement2 [T
    = AUTH2_M::SYSTEM_2
```

```
No attack found

Checking assertion STOP [T= SYSTEM\diff(Events,{|INTRUDER_M::verify|})
No attack found

Done
```

A.2 PERMIS

```
Starting FDR
Checking /home/madtosh/Documents/03_Research/03_CU/03_P2PFileSystem/03_Casper-
    FDR/02_Permis/Final/Permis1of3.csp

Checking assertion SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
No attack found

Checking assertion SECRET_M::SEQ_SECRET_SPEC [T= SECRET_M::SYSTEM_S_SEQ
No attack found

Checking assertion AUTH1_M::AuthenticateSERVERToRESPONDERAgreement [T= AUTH1_M
    ::SYSTEM_1
No attack found

Done

Starting FDR
Checking /home/madtosh/Documents/03_Research/03_CU/03_P2PFileSystem/03_Casper-
    FDR/02_Permis/Final/Permis2of3.csp

Checking assertion SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
No attack found

Checking assertion SECRET_M::SEQ_SECRET_SPEC [T= SECRET_M::SYSTEM_S_SEQ
No attack found

Done
```

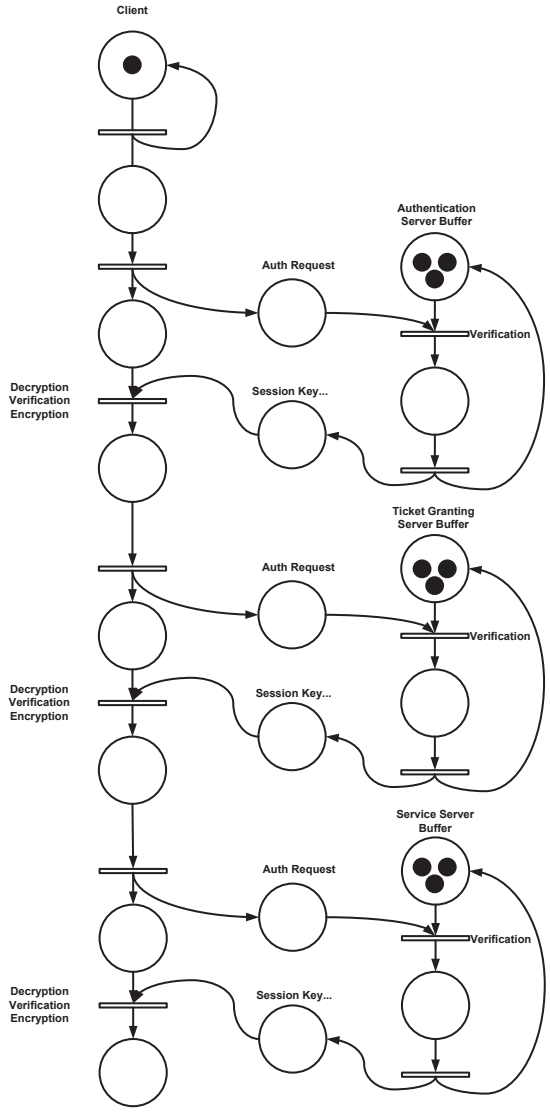
```
Starting FDR
Checking /home/madtosh/Documents/03_Research/03_CU/03_P2PFileSystem/03_Casper-
    FDR/02_Permis/Final/Permis3of3.csp

Checking assertion SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
No attack found

Checking assertion SECRET_M::SEQ_SECRET_SPEC [T= SECRET_M::SYSTEM_S_SEQ
No attack found

Done
```

A.3 Shibboleth

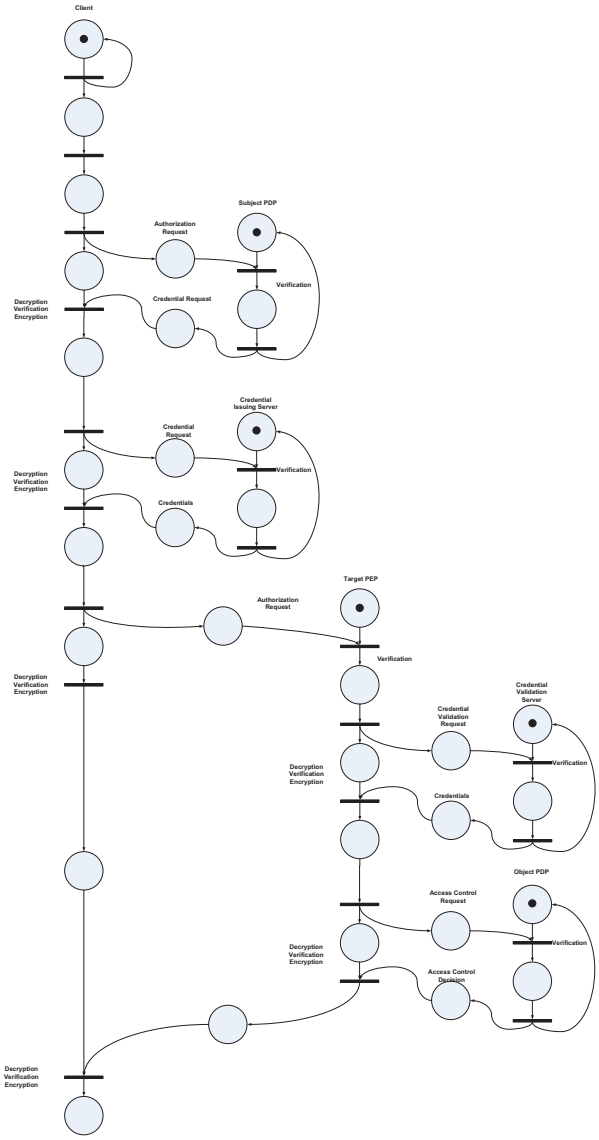
```
Starting FDR
Checking /home/madtosh/Documents/03_Research/03_CU/03_P2PFileSystem/03_Casper-
    FDR/03_Shibboleth/Shibboleth.csp

Checking assertion SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
No attack found

Checking assertion SECRET_M::SEQ_SECRET_SPEC [T= SECRET_M::SYSTEM_S_SEQ
No attack found

Checking assertion AUTH1_M::AuthenticateRESPONDERToSERVERAgreement [T= AUTH1_M
    ::SYSTEM_1
No attack found

Checking assertion AUTH2_M::AuthenticateSERVERToRESPONDERAgreement [T= AUTH2_M
    ::SYSTEM_2
No attack found

Done
```

Appendix B Petri Net Graphs

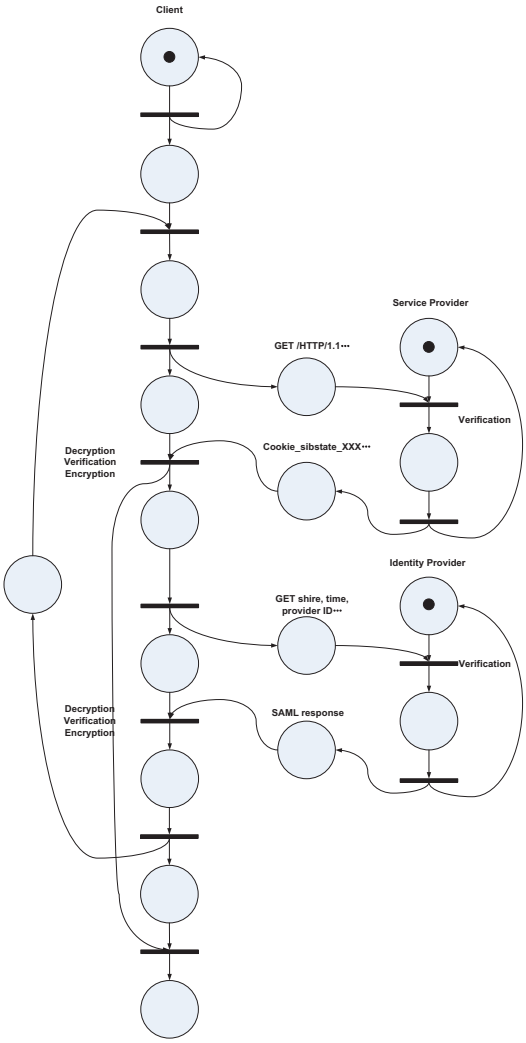
B.1 Kerberos



B.2 PERMIS


Client
Subject PDP
Authorization Request
Verification
Credential Request
Decryption Verification Encryption
Credential Issuing Server
Credential Request
Verification
Credentials
Decryption Verification Encryption
Target PEP
Authorization Request
Verification
Decryption Verification Encryption
Credential Validation Server
Credential Validation Request
Verification
Decryption Verification Encryption
Credentials
Object PDP
Access Control Request
Verification
Decryption Verification Encryption
Access Control Decision
Decryption Verification Encryption

B.3 Shibboleth



Appendix C Karp-Miller Trees Graphs

C.1 Kerberos

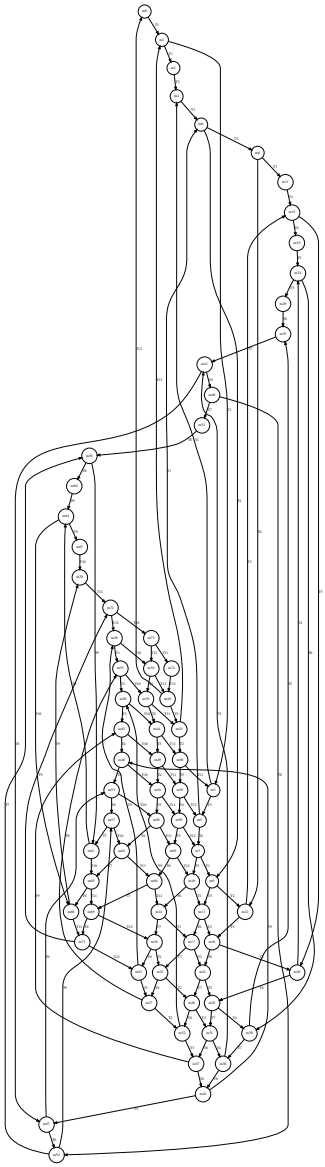


Figure 1: Karp-Miller Tree of Kerberos with Two initial tokens

C.2 PERMIS

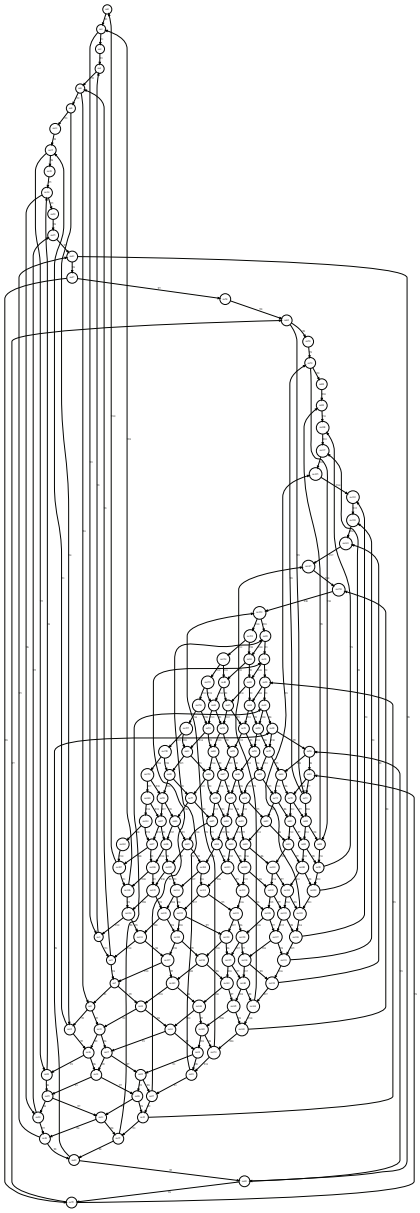


Figure 2: Karp-Miller Tree of PERMIS with Two initial tokens

C.3 Shibboleth

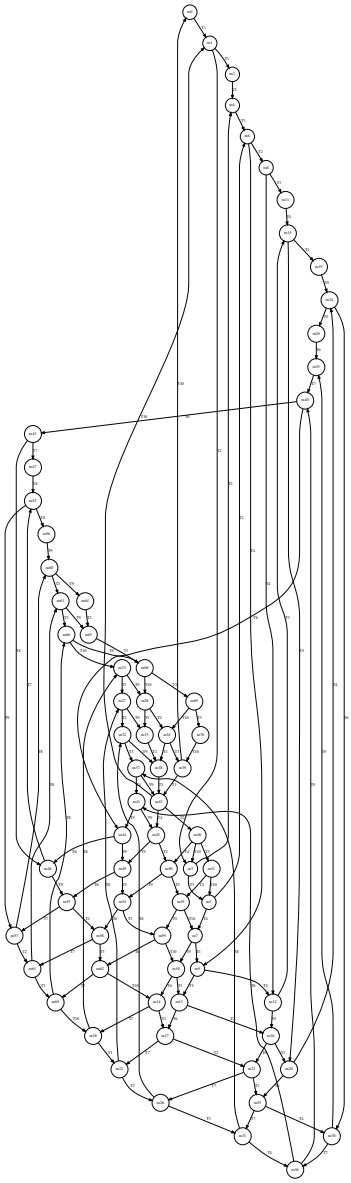


Figure 3: Karp-Miller Tree of Shibboleth with Two initial tokens

Appendix D Casper Scripts

D.1 Kerberos (1 of 2)

```
-- Kerberos 5 protocol (part 1/2) by Seok B. Yun
-------------------------

-- Purpose : keep secracy between AS<->TGS<->DS

-- Abbreviations
-------------------
-- C       : client
-- DS      : Data Server
-- TGS     : Ticket Granting Server
-- AS      : Authentication Server
-- Addr    : address (not used)
-- Life    : lifetime of ticket (not used)
-- Key(x)  : x's private key
-- Kxy     : Session Key for x and y
-- Txy     : x's ticket to use y
-- Ax      : Authenticator for x

-- Ticket Definition
-----------------------
--    {DS, C, Addr, timestamp, life, Ksc}Key(DS)

-- Authenticator Definition
-----------------------------
--    {C, Addr, timestamp}Ksc



-------------------
-- CODE STARTS --
-------------------


#Protocol description
```

```
-- 1. Getting TGS ticket (C <-> AS)
-------------------------------------

0.   -> C   : AS, TGS
1. C -> AS  : C, TGS
2. AS -> C  : {C, TGS, ts1, kct}{SKey(TGS)} % Tct
3. AS -> C  : {kct}{SKey(C)}

-- 2. Requesting a Service tickets (C -> TGS)
-----------------------------------------

4. C -> TGS : Tct % {C, TGS, ts1, kct}{SKey(TGS)}
[ts1==now or ts1+1==now]
5. C -> TGS : {C, ts2}{kct}
[ts2==now or ts2+1==now]



#Free variables
C, AS, TGS : Agent
SKey: Agent -> SecretKey
kct : SessionKey
ts1, ts2: TimeStamp
--va : Nonce

InverseKeys = (SKey, SKey), (kct, kct)

#Processes
INITIATOR(C) knows SKey(C)
RESPONDER(TGS) knows SKey(TGS)
SERVER(AS) knows SKey(C), SKey(TGS) generates kct

#Specification
Agreement(C, AS, [])
Agreement(AS, C, [])
TimedAgreement(C, TGS, 2, [])
```

```
#Actual variables
Client, AuthServer, TGServer, Mallory : Agent
Kct : SessionKey
--Va : Nonce
TimeStamp = 0 .. 3
MaxRunTime = 2
InverseKeys = (Kct, Kct)

#Functions
symbolic SKey

#System
INITIATOR(Client)
RESPONDER(TGServer)
RESPONDER(AuthServer)

#Intruder Information
Intruder = Mallory
IntruderKnowledge = {Client, Mallory, AuthServer, SKey(Mallory)}
Guessable = SessionKey
Crackable = SessionKey
Crackable = SecretKey
Crackable = Password
```

D.2 Kerberos (2 of 2)

```
-- Kerberos 5 protocol (part 2/2) by Seok B. Yun
------------------------

-- Purpose : keep secracy between AS<->TGS<->DS

-- Abbreviations
-----------------
-- C       : client
-- DS      : Data Server
-- TGS     : Ticket Granting Server
```

```
-- AS      : Authentication Server
-- Addr    : address (not used)
-- Life    : lifetime of ticket (not used)
-- Key(x)  : x's private key
-- Kxy     : Session Key for x and y
-- Txy     : x's ticket to use y
-- Ax      : Authenticator for x

-- Ticket Definition
----------------------

--    {DS, C, Addr, timestamp, life, Ksc}Key(DS)

-- Authenticator Definition
-----------------------------

--    {C, Addr, timestamp}Ksc



-------------------
-- CODE STARTS --
-------------------

#Protocol description
-- 3. Getting Service Tickets (C <- TGS)
---------------------------------------

0.    -> C    : kct, DS
1.    -> TGS  : C, DS, kct
3. TGS -> C : {C, DS, ts1, kcs}{SKey(DS)} % Tcs
4. TGS -> C : {kcs}{kct}

-- 4. Requesting a Data Service
---------------------------------

5. C -> DS : Tcs % {C, DS, ts1, kcs}{SKey(DS)}
[ts1==now or ts1+1==now]
6. C -> DS : {C, ts2}{kcs}
```

```
[ts2==now or ts2+1==now]

#Free variables
C, TGS, DS : Agent
SKey: Agent -> SecretKey
kct, kcs : SessionKey
ts1, ts2: TimeStamp

InverseKeys = (SKey, SKey), (kct, kct), (kcs, kcs)

#Processes
INITIATOR(C) knows SKey(C)
RESPONDER(DS) knows SKey(DS)
SERVER(TGS) knows SKey(DS) generates kcs

#Specification
Agreement(TGS, C, [])
TimedAgreement(C, DS, 2, [])

#Actual variables
Client, TGServer, DataServer, Mallory : Agent
Kct, Kcs : SessionKey
TimeStamp = 0 .. 3
MaxRunTime = 2
InverseKeys = (Kct, Kct), (Kcs, Kcs)

#Functions
symbolic SKey

#System
INITIATOR(Client)
RESPONDER(DataServer)
RESPONDER(TGServer)
```

```
#Intruder Information
Intruder = Mallory
IntruderKnowledge = {Client, Mallory, TGServer, DataServer, SKey(Mallory)}
Guessable = SessionKey
Crackable = SessionKey
Crackable = SecretKey
Crackable = Password
```

D.3 PERMIS (1 of 3)

```
-- Permis Authorization protocol (part 1/3) by Seok B. Yun
-------------------------

-- Purpose : keep secracy between SPDP<->CIS
-- All protocol should be undertaken by TLS/SSL


-- Abbreviations
-----------------

-- CPEP       : Client PEP
-- SPDP       : Subject PDP
-- CIS        : Credential Issuing Server

-- TPEP       : Target PEP

-- rsc        : resource field (Attribute id, value, data type)
-- wst        : WS-TRUST token

-- PK(x)      : x's public key
-- SK(x)      : x's secret key



-----------------

-- CODE STARTS --

-----------------

#Protocol description
-- 1. Authorization Request (CPEP <-> SPDP)
```

```
-------------------------------------

0.       -> CPEP : SPDP
1. CPEP -> SPDP : {SPDP, rsc}{PK(SPDP)}
2.       -> SPDP : CIS
3. SPDP -> CPEP : {rsc, SPDP, f(rsc, SPDP)}{PK(CIS)} % cred1, CIS

-- 2. Credential Request (CPEP <-> CIS)
-------------------------------------

4. CPEP -> CIS    : cred1 % {rsc, SPDP, f(rsc, SPDP)}{PK(CIS)}
5.       -> CIS    : wst, TPEP
6. CIS  -> CPEP   : {f(wst, CIS)}{PK(TPEP)} % wstrusthash

#Free variables
CPEP, SPDP, CIS, TPEP: Agent
f: HashFunction
rsc, wst : Nonce
PK: Agent -> PublicKey
SK: Agent -> SecretKey
InverseKeys = (PK, SK)

#Processes
INITIATOR(CPEP, rsc) knows PK, SK(CPEP)
RESPONDER(CIS, wst) knows PK, SK(CIS)
SERVER(SPDP) knows PK, SK(SPDP)

#Specification
Secret(SPDP,rsc,[CIS, CPEP])
Secret(CIS,wst,[SPDP, CPEP])

-- Agreement between Subject PDP and Credential Issuing Server
-----------------------------------------------------------

Agreement(SPDP, CIS,[])

#Actual variables
```

```
ClientPEP, SubjectPDP, CredIS, Mallory: Agent
Rsc, Wst: Nonce

#Functions
symbolic PK, SK

#System
INITIATOR(ClientPEP, Rsc)
RESPONDER(CredIS, Wst)
SERVER(SubjectPDP)

#Intruder Information
Intruder=Mallory
IntruderKnowledge={ClientPEP, SubjectPDP, CredIS, Mallory}
```

D.4 PERMIS (2 of 3)

```
-- Permis Authorization protocol (part 2/3) by Seok B. Yun
-------------------------

-- Purpose : keep secracy between CPEP<->CVS
-- All protocol should be undertaken by TLS/SSL

-- Abbreviations
-----------------

-- CPEP       : Client PEP
-- CVS        : Credential Validation Server
-- TPEP       : Target PEP

-- rsc        : resource field (Attribute id, value, data type)
-- wst        : WS-TRUST token

-- PK(x)      : x's public key
-- SK(x)      : x's secret key
```

```
------------------
-- CODE STARTS --
------------------

#Protocol description
-- 3. Authorization Request (CPEP <-> TPEP)
-----------------------------------------

0.       -> CPEP : TPEP, rsc, wst, CIS, CVS, f(wst, CIS)
1. CPEP -> TPEP : {rsc, CPEP}{PK(TPEP)}
2. CPEP -> TPEP : {f(wst, CIS)}{PK(CVS)} % wstrusthash

-- 4. Credential Request (TPEP <-> CVS)
-----------------------------------------

3.       -> TPEP : CVS
4.       -> CVS  : wst, CIS, na
5. TPEP -> CVS  : wstrusthash % {f(wst, CIS)}{PK(CVS)}
7. CVS -> TPEP  : {na}{PK(TPEP)}

#Free variables
CPEP, TPEP, CVS, CIS : Agent
f: HashFunction
rsc, wst, na: Nonce
PK: Agent -> PublicKey
SK: Agent -> SecretKey
InverseKeys = (PK, SK)

#Processes
INITIATOR(CPEP) knows PK, SK(CPEP)
RESPONDER(CVS) knows PK, SK(CVS)
SERVER(TPEP) knows PK, SK(TPEP)

#Specification
Secret(CPEP, rsc, [TPEP])
Secret(CPEP, wst, [CVS])
```

```
#Actual variables
ClientPEP, TargetPEP, CredVS, CredIS, Mallory: Agent
Rsc, Wst, Na: Nonce

#Functions
symbolic PK, SK

#System
INITIATOR(ClientPEP)
RESPONDER(CredVS)
SERVER(TargetPEP)

#Intruder Information
Intruder=Mallory
IntruderKnowledge={ClientPEP, TargetPEP, CredVS, Mallory}
```

D.5 PERMIS (3 of 3)

```
-- Permis Authorization protocol (part 2/3) by Seok B. Yun
-------------------------

-- Purpose : keep secracy between CPEP<->CVS
-- All protocol should be undertaken by TLS/SSL

-- Abbreviations
-----------------

-- CPEP       : Client PEP
-- TPEP       : Target PEP
-- OPDP       : Object PDP

-- rsc        : resource field (Attribute id, value, data type)
-- wst        : WS-TRUST token

-- PK(x)      : x's public key
-- SK(x)      : x's secret key
```

```
-------------------
-- CODE STARTS --
-------------------

#Protocol description

-- 5. Service Request (CPEP -> TPEP -> OPDP)
-----------------------------------------

0.       -> CPEP : TPEP, rsc
1. CPEP -> TPEP : {rsc, CPEP}{PK(TPEP)}
2.       -> TPEP : OPDP
3.       -> OPDP : na
4. TPEP -> OPDP : {rsc, CPEP}{PK(OPDP)}

-- 6. Service Enable (CPEP -> TPEP -> OPDP)
-----------------------------------------

5. OPDP -> TPEP : {na, CPEP}{PK(TPEP)}
6. TPEP -> CPEP : {na}{PK(CPEP)}

#Free variables
CPEP, TPEP, OPDP : Agent
rsc, na: Nonce
PK: Agent -> PublicKey
SK: Agent -> SecretKey
InverseKeys = (PK, SK)

#Processes
INITIATOR(CPEP) knows PK, SK(CPEP)
RESPONDER(OPDP, rsc) knows PK, SK(OPDP)
SERVER(TPEP) knows PK, SK(TPEP)

#Specification
Secret(OPDP, rsc, [TPEP, CPEP])
Secret(OPDP, na, [CPEP, TPEP])
```

```
#Actual variables
ClientPEP, TargetPEP, ObjectPDP, Mallory: Agent
Rsc, Na: Nonce

#Functions
symbolic PK, SK

#System
INITIATOR(ClientPEP)
RESPONDER(ObjectPDP, Rsc)
SERVER(TargetPEP)

#Intruder Information
Intruder=Mallory
IntruderKnowledge={ClientPEP, TargetPEP, ObjectPDP, Mallory}
```

D.6 Shibboleth

```
-- Shibboleth protocol by Seok B. Yun
-------------------------

-- Just cookie is used for all.

-- Abbreviations
-------------------

-- C          : Client
-- SP         : Service Provider
-- IP         : Identity Provider

-- PK(x)      : x's public key
-- SK(x)      : x's secret key



--------------------

-- CODE STARTS --
```

```
#Protocol description
0.   -> C  : SP
1. C -> SP : {SP, na}{PK(SP)}
2.   -> SP : IP
3. SP -> C : {na, SP, f(na, SP)}{PK(IP)} % cookie1, IP
4. C -> IP : cookie1 % {na, SP, f(na, SP)}{PK(IP)}
5.   -> IP : nb
6. IP -> C : {nb, IP, f(nb, IP)}{PK(SP)} % cookie2
7. C -> SP : cookie2 % {nb, IP, f(nb, IP)}{PK(SP)}

#Free variables
C, SP, IP: Agent
f: HashFunction
na, nb : Nonce
PK: Agent -> PublicKey
SK: Agent -> SecretKey
InverseKeys = (PK, SK)

#Processes
INITIATOR(C, na) knows PK, SK(C)
RESPONDER(SP, nb) knows PK, SK(SP)
SERVER(IP) knows PK, SK(IP)

#Specification
Secret(SP,na,[IP])
Secret(IP,nb,[SP])
Agreement(SP, IP,[])
Agreement(IP, SP,[])

#Actual variables
Client, ServiceProvider, IdentityProvider, Mallory: Agent
Na, Nb: Nonce
```

```
#Functions
symbolic PK, SK

#System
INITIATOR(Client, Na)
RESPONDER(ServiceProvider, Nb)
SERVER(IdentityProvider)

#Intruder Information
Intruder=Mallory
IntruderKnowledge={Client, ServiceProvider, IdentityProvider, Mallory}
```

Bibliography

[1] Failure divergence refinement 2 (http://www.fsel.com/).

[2] Freshmeat: Release summaries of parallel virtual file system version 2.7.1 (http://freshmeat.net/projects/pvfs).

[3] Gpfs v3.1administration and programming reference.

[4] Differences between saml 2.0 and 1.1(http://saml.xml.org/differences-between-saml-2-0-and-1-1), jan 2008.

[5] ISO/IEC TR 14516:2002. *ISO/IEC TR 14516:2002, Information technology – Security techniques – Guidelines for the use and management of Trusted Third Party services.* ISO, Geneva, Switzerland, 2002.

[6] Scott W. Ambler. Uml 2 sequence diagram (http://www.agilemodeling.com/artifacts/sequencediagram.htm).

[7] Donald Bell. Uml basics: The sequence diagram (http://www.ibm.com/developerworks/rational/library/3101.html), February 2004.

[8] Matt Blaze, Matt Blaze, Joan Feigenbaum, and Jack" Lacy. Decentralized trust management. *IN PROCEEDINGS OF THE 1996 IEEE SYMPOSIUM ON SECURITY AND PRIVACY*, pages 164–173, 1996.

[9] Hong bo Xie, Yuan chen Wu, and Ming tian Zhou. A directed graph-based authentication protocol model and its security analysis. In *Frontier of Computer Science and Technology, 2006. FCST '06. Japan-China Joint Workshop on*, pages 89 –96, nov. 2006.

[10] R. R Brooks. *Disruptive security technologies with mobile code and peer-to-peer networks.* CRC Press, Boca Raton, 2004.

[11] G. Caronni. Walking the web of trust. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2000. (WET ICE 2000). Proeedings. IEEE 9th International Workshops on*, pages 153 –158, 2000.

[12] D. W. Chadwick. Modular permis project: What is permis? (http://sec.cs.kent.ac.uk/permis/documents/concept.shtml), July 2011.

[13] David Chadwick. Functional components of grid service provider authorisation service. *MIDDLEWARE", OGF GWD-I*, nov. 2009.

[14] David Chadwick, Gansen Zhao, Sassa Otenko, Romain Laborde, Linying Su, and Tuan Anh Nguyen. Permis: a modular authorization infrastructure. *Concurrency and Computation: Practice and Experience*, 20(11):1341–1357, 2008.

[15] David W. Chadwick and Alexander Otenko. The permis x.509 role based privilege management infrastructure. *Future Generation Computer Systems*, 19(2):277 – 289, 2003.

[16] Li Chen and Mingxia Shi. Security analysis and improvement of yahalom protocol. In *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, pages 1137 –1140, june 2008.

[17] Li Chen and Weixian Wang. An improved nssk protocol and its security analysis based on logic approach. In *Communications, Circuits and Systems, 2008. ICCCAS 2008. International Conference on*, pages 772 –775, may 2008.

[18] T. Coffey and P. Saidha. Logic for verifying public-key cryptographic protocols. *Computers and Digital Techniques, IEE Proceedings* -, 144(1):28 –32, jan 1997.

[19] R. Corin and A. Saptawijaya. A logic for constraint-based security protocol analysis. In *Security and Privacy, 2006 IEEE Symposium on*, pages 14 pp. –168, may 2006.

[20] R David and H Alla. *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*. Prince Hall, 1992.

[21] Linying Su David Chadwick. Use of ws-trust and saml to access a credential validation service. *MIDDLEWARE", OGF GWD-I*, nov. 2009.

[22] Romain Laborde David Chadwick, Linying Su. Use of xacml request context to obtain an authorisation decision. *MIDDLEWARE", OGF GWD-I*, nov. 2009.

[23] Juan Deng. *Connected Vehicle Information Assurance*. Phd dissertation, Clemson University, Clemson SC 29634, August 2011.

[24] Marlena Erdos, Marlena Erdos, and Scott" Cantor. Shibboleth-architecture draft v05 draft-internet2-shibboleth-architecture-05.html. 2002.

[25] Kai Fan, Hui Li, and Yue Wang. Security analysis of the kerberos protocol using ban logic. In *Information Assurance and Security, 2009. IAS '09. Fifth International Conference on*, volume 2, pages 467 –470, aug. 2009.

[26] Gombás Gábor, Gombás Gábor, and Témavezető Frohner" Ákos. Evaluation of distributed authentication, authorization and directory services.

[27] Jill Gemmill, John-Paul Robinson, Tom Scavo, and Purushotham Bangalore. Cross-domain authorization for federated virtual organizations using the myvocs collaboration environment. *Concurrency and Computation: Practice and Experience*, 21(4):509–532, 2009.

[28] C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[29] Wei Jie, Junaid Arshad, Richard Sinnott, Paul Townend, and Zhou Lei. A review of grid authentication and authorization technologies and support for federated access control. *ACM Comput. Surv.*, 43(2):12:1–12:26, February 2011.

[30] Richard R. Brooks Juan Deng. From csp to petri net: A complete security protocol analysis profile.

[31] Richard M. Karp and Raymond E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147 – 195, 1969.

[32] G. Lowe. Casper: a compiler for the analysis of security protocols. In *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, pages 18 –30, jun 1997.

[33] G. Lowe and B. Roscoe. Using csp to detect errors in the tmn protocol. *Software Engineering, IEEE Transactions on*, 23(10):659 –669, oct 1997.

[34] Gavin Lowe. Casper: A compiler for the analysis of security protocols.

[35] W. Mao and C. Boyd. Towards formal analysis of security protocols. In *Computer Security Foundations Workshop VI, 1993. Proceedings*, pages 147 –158, jun 1993.

[36] David E. Martin. Shibboleth : An open source, federated single sign-on system. martinde@northwestern.edu, mar 2009.

[37] C. Metz. Aaa protocols: authentication, authorization, and accounting for the internet. *Internet Computing, IEEE*, 3(6):75 –79, nov/dec 1999.

[38] Sun Microsystems. Designing an xml data structure.

[39] S. P. Miller, S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H." Saltzer. Kerberos authentication and authorization system. *IN PROJECT ATHENA TECHNICAL PLAN*, 1987.

[40] Y. Nakamura, M. Tatsubori, T. Imamura, and K. Ono. Model-driven security based on a web services security architecture. In *Services Computing, 2005 IEEE International Conference on*, volume 1, pages 7 – 15 vol.1, july 2005.

[41] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978.

[42] Mark Needleman. The shibboleth authentication/authorization system. *Serials Review*, 30(3):252 – 253, 2004.

[43] B.C. Neuman and T. Ts'o. Kerberos: an authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33 –38, sep 1994.

[44] OASIS. Oasis security services use cases and requirements (http://www.oasis-open.org/committees/security/docs/draft-sstc-use-strawman-03.html), February 2001.

[45] R.A. Oldfield, L. Ward, R. Riesen, A.B. Maccabe, P. Widener, and T. Kordenbrock. Lightweight i/o for scientific applications. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1 –11, sept. 2006.

[46] R.A. Oldfield, P. Widener, A.B. Maccabe, L. Ward, and T. Kordenbrock. Efficient data-movement for lightweight i/o. In *Cluster Computing, 2006 IEEE International Conference on*, pages 1 –9, sept. 2006.

[47] A. W. Roscoe. *The theory and practice of concurrency.* Prentice Hall, 1998. The text book teaching material can be found at `http://www.comlab.ox.ac.uk/publications/books/concurrency/`.

[48] A.W. Roscoe. Modelling and verifying key-exchange protocols using csp and fdr. In *Computer Security Foundations Workshop, 1995. Proceedings., Eighth IEEE*, pages 98 –107, jun 1995.

[49] P. Ryan and S. Schneider. *The modelling and analysis of security protocols: the csp approach.* Addison-Wesley Professional, first edition, 2000.

[50] Frank Schmuck and Roger Haskin. Gpfs: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[51] S. Schneider. Security properties and csp. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 174 –187, may 1996.

[52] Steve Schneider. *Concurrent and real-time systems: the CSP approach.* Worldwide series in computer science. Wiley, Chichester, 2000.

[53] H.R. Shahriari and R. Jalili. Using csp to model and analyze transmission control protocol vulnerabilities within the broadcast network. In *Networking and Communication Conference, 2004. INCC 2004. International*, pages 42 – 47, june 2004.

[54] Frederick T. Sheldon. Analysis of real-time concurrent system models based on csp using stochastic petri nets. In *Proceedings of the 12th European Simulation Multiconference on Simulation - Past, Present and Future*, pages 776–783. SCS Europe, 1998.

[55] T. Srivatanakul, J.A. Clark, S. Stepney, and F. Polack. Challenging formal specifications by mutation: a csp security example. In *Software Engineering Conference, 2003. Tenth Asia-Pacific*, pages 340 – 350, dec. 2003.

[56] Jennifer G Steiner, Clifford Neuman, and Jeffrey I Schiller. Kerberos: An authentication service for open network systems. *Winter 1988 USENIX Conference*, pages 191–202, 1988.

[57] Li Tingyuan, Liu Xiaodong, Qin Zhiguang, and Zhang Xuanfang. Formal analysis for security of otway-rees protocol with ban logic. In *Database Technology and Applications, 2009 First International Workshop on*, pages 590 –593, april 2009.

[58] Li Tingyuan, Liu Xiaodong, Qin Zhiguang, and Zhang Xuanfang. An improved security protocol formal analysis with ban logic. In *Electronic Commerce and Business Intelligence, 2009. ECBI 2009. International Conference on*, pages 102 –105, june 2009.

[59] Tom Scavo Valerio Venturi. Use of saml to retrieve authorization credentials. *MIDDLEWARE", OGF GWD-I*, nov. 2009.

[60] Hao Yin, Hao Yin, Sofia Brenes Barahona, Donald F. Mcmullen, Marlon Pierce, Kianosh Huffman, and Geoffrey" Fox. A permis-based authorization solution between portlets and back-end web services.